

The Algebra of Recursive Graph Transformation Language UnCAL: Complete Axiomatisation and Iteration Categorical Semantics

Makoto Hamana¹, Kazutaka Matsuda² and Kazuyuki Asada³

¹ *Department of Computer Science, Gunma University, Japan*

² *Graduate School of Information Sciences, Tohoku University, Japan*

³ *Department of Computer Science, University of Tokyo, Japan*

The aim of this paper is to provide mathematical foundations of a graph transformation language, called UnCAL, using categorical semantics of type theory and fixed points. About twenty years ago, Buneman et al. developed a graph database query language UnQL on the top of a functional meta-language UnCAL for describing and manipulating graphs. Recently, the functional programming community has shown renewed interest in UnCAL, because it provides an efficient graph transformation language which is useful for various applications, such as bidirectional computation.

In order to make UnCAL more flexible and fruitful for further extensions and applications, in this paper, we give a more conceptual understanding of UnCAL using categorical semantics. Our general interest of this paper is to clarify what is the algebra of UnCAL. Thus, we give an equational axiomatisation and categorical semantics of UnCAL, both of which are new. We show that the axiomatisation is complete for the original bisimulation semantics of UnCAL. Moreover, we provide a clean characterisation of the computation mechanism of UnCAL called “structural recursion on graphs” using our categorical semantics. We show a concrete model of UnCAL given by the λ G-calculus, which shows an interesting connection to lazy functional programming.

Contents

1	Introduction	2
1.1	Why the algebra of UnCAL? — models of graphs, bisimulation and structural recursion	3
1.2	Bridge among database, categorical semantics, and functional programming	4
1.3	UnCAL overview	5
1.4	Related work	8
1.5	Organisation	9
2	UnCAL and its Equational Theory	9
2.1	Syntax	10
2.2	Typed syntax	10
2.3	Equational axiomatisation	13

3	Categorical Semantics	19
3.1	Interpretation	20
3.2	Categorical completeness	22
4	Characterisation of Structural and Primitive Recursion	24
4.1	Overview	24
4.2	Structural recursion on graphs	26
4.3	Primitive recursion on graphs	29
4.4	Application to the fusion law	32
5	Completeness for Bisimulation	33
5.1	Equational logic for μ -terms	34
5.2	Characterising UnCAL normal forms	35
5.3	Completeness of the axioms for bisimulation	38
6	Instances of UnCAL Models	38
6.1	The bisimulation model	39
6.2	A CPO model	39
6.3	The λ G-calculus model	40
7	Discussion: Variations of UnCAL and Related Graph Calculi	45
	Appendix A The Graph Model of UnCAL	50
	A.1 UnCAL graphs	50
	A.2 The category of UnCAL graphs and a model	51

1. Introduction

Graph databases, which store graphs rather than relations in ordinary relational database, are used as back-ends of various web and net services. Therefore it is one of the important software systems in the Internet society. About twenty years ago, Buneman et al. (1996, 1997, 2000) developed a graph database query language UnQL (Unstructured data Query Language) on top of a functional meta-language **UnCAL** (Unstructured Calculus) for describing and manipulating graph data. The term “unstructured” refers to unstructured or semi-structured data, i.e., data having no assumed format in a database (in contrast to relational database). Recently, the functional programming community has expressed renewed interest in UnCAL. A research group in Tokyo discovered a new application area of UnCAL in so-called bidirectional transformations on graph data (Hidaka et al., 2010, 2011, 2013a; Asada et al., 2013), because it provides an efficient graph transformation language. It is also suitable for various applications, including software engineering (Yu et al., 2012; Hidaka et al., 2013b). UnCAL has been extended and refined in various directions. Those developments have enhanced the importance of UnCAL.

In order to make UnCAL more flexible and fruitful for further extensions and applications, in this paper, we present a more conceptual understanding of UnCAL using semantics of type theory and fixed points. Our general interest in this study is clarification of the *algebra of UnCAL*. Therefore, we give an equational axiomatisation and categorical semantics of UnCAL, both of which are new. We show that the axiomatisation exactly matches with the original formulation. That is, it is complete for the original bisimulation of UnCAL graphs. Moreover, we provide a clean characterisation

of the computation mechanism of UnCAL called “structural recursion on graphs” using our categorical semantics.

1.1. *Why the algebra of UnCAL? — models of graphs, bisimulation and structural recursion*

Besides the practical importance of UnCAL, this study is also conceptually and theoretically motivated. UnCAL deals with graphs *modulo bisimulation*, rather than isomorphisms or strict equality. Numerous studies have examined *algebraic* or *categorical characterisation* of graphs (or DAGs) *modulo isomorphism* in the field of semantics of programming languages, such as (Ariola and Klop, 1996; Ariola and Blom, 1997; Hasegawa, 1997b,a; Corradini and Gadducci, 1999; Milner, 1996, 2005, 2006; Hamana, 2010, 2012; Gibbons, 1995; Fiore and Campos, 2013). Algebraic characterisation is associated directly to the constructive nature of graphs. Thereby these can be applied nicely to functional programming concepts such as the datatype of graphs and its “fold” or structural recursion (Gibbons, 1995; Hamana, 2010) and λ -calculus (Ariola and Blom, 1997; Hasegawa, 1997b,a). In this respect, algebraic characterisation has both theoretical benefits (e.g. new semantical structures) and practical benefits (e.g. programming concepts).

Graphs *modulo bisimulation* have mainly been studied in the context of concurrency (Milner, 1984, 1989), rather than functional programming and program semantics. In concurrency, graphs are used to represent traces (or behavior) of processes, and bisimulation are used to compare them. Semantic tools to model them are often coalgebraic because a graph can be regarded as a coalgebraic structure that outputs node information along its edges, e.g. (Aczel et al., 2003; Ghani et al., 2005). Then bisimulation is modelled as a relation on coalgebras, e.g. (Staton, 2011).

The design of UnCAL is unique from the viewpoint of existing works related to semantics of graphs, bisimulation, and programming described above. In fact, UnCAL is similar to functional programming (because of computation by “structural recursion on graphs”), but the basic data structure is non-standard, i.e. graphs *modulo bisimulation*. One reason for this design choice of UnCAL is due to efficiency. Therefore, the original UnCAL’s formulation is mainly graph algorithmic. Basic graph theory, algorithmic evaluation, and ordinary relational treatment of bisimulation are the main tools.

Clarifying more mathematical semantics of UnCAL is challenging. Three key concepts of UnCAL — graphs, bisimulation, and functional programming — have been modelled in different settings as described above, but they have not been examined in a single framework. Consequently, our central question is

What is the mathematical structure for modelling UnCAL?

Finding the structure has additional importance because it tightly connects to the notion of structural recursion in UnCAL. In principle, a structural recursive function is a mapping that preserves “structures”. However, such structures of UnCAL have not been pursued seriously and have remained vague in the original and subsequent developments of UnCAL.

Our approach to tackle this problem is algebraic and categorical, rather than a coalgebraic approach. The reason is described below. Our central idea is to follow the methodology of modelling structural recursion or the “fold” of algebraic datatype by the universality of datatypes in functional programming, e.g. (Hagino, 1987). It is the established methodology to characterise datatypes and recursion in a single setting, where the general structure is functor-algebra, and datatype is characterised as an initial functor-algebra. Structural recursion follows automatically from the universality. This methodology can be achieved only using an algebraic approach. Therefore, the coalgebraic approach is unsuitable for our purposes.

In the case of UnCAL, our methodology is

- (1) to identify a suitable category for UnCAL, which has all required constructs and properties, and
- (2) to characterise the “datatype” of UnCAL graphs as a universal one in it.

Of course, these are not merely an algebraic datatype, functor-algebras and an initial algebra, unlike (Hamana, 2010), which modelled rooted graphs without any quotient by an initial functor-algebra in a presheaf category. It is necessary to seek more involved and suitable mathematical structure.

As described herein, we identify the mathematical structure of UnCAL to be *iteration categories* (Ésik, 1999a; Bloom and Ésik, 1993) and our notion of *L-monoids* (Def. 3.2). We characterise UnCAL’s graphs (regarded as the “datatype”) by the universal property of certain *L-monoid* and iteration category. Structural recursion follows from the universality.

This universal characterisation of UnCAL shows that UnCAL has a standard status, i.e., it is not an ad-hoc language of graphs transformation. The simply-typed λ -calculus is a standard functional language because it is universal in cartesian closed categories. The universal characterisation of UnCAL also provides syntax-independent formulation. Therefore, the current particular syntax of UnCAL is unimportant. What is most important we think is that the necessary mathematical structures for a language of graph transformation with bisimulation and structural recursion are iteration categories and *L-monoids*. In this respect, UnCAL is one such language, but it is also *the most standard language* because it has exactly these required structures for graph transformation.

1.2. Bridge among database, categorical semantics, and functional programming

This universal characterisation provides another benefit. We mentioned that the current syntax of UnCAL is unimportant. We will actually give another syntax of UnCAL using a λ -calculus, called the λ G-calculus (§6.3). It is because the syntax and equational theory of the λ G-calculus forms a categorical model of UnCAL, Therefore, by universality, there exists a structure preserving translation from the universal one, i.e. the original UnCAL syntax, to the λ G-calculus. The translation is also invertible (Prop. 6.3). Therefore, one can use the λ G-calculus as an alternative syntax, computation and reasoning method of UnCAL. Moreover, an efficient abstract machine for λ G-calculus has been implemented as a functional language FUnCAL (Matsuda and Asada, 2015).

UnCAL originated in the field of database, but its language design also gained from ideas in concurrency and functional programming, as mentioned recently in Buneman’s retrospect on the relationship between database and programming languages (Buneman, 2015). Our categorical approach *sharpens* this line of good relationship between database and programming languages, not only at the level of language design and implementations, but also at the level of rigorous mathematical semantics. It bridges different areas of computer science, i.e. graph database (UnCAL), categorical semantics (iteration category), and functional programming (the λ G-calculus) by categorical interpretation.

1.3. UnCAL overview

1.3.1. We begin by introducing UnCAL. UnCAL deals with graphs. Hence, it is better to start with viewing how concrete semi-structured data is processed in UnCAL. Consider the semi-structured data **sd** below which is taken from (Buneman et al., 2000). It contains information about country, e.g. geography, people, government, etc. It is depicted as a tree

above, in which edges and leaves are labelled. Using UnCAL’s term language for describing graphs (and trees), this is defined by **sd** shown at the right. Then we can define functions in

```
sd <- country:{name:"Luxembourg",
  geography:{coordinates:{long:"49 45N", lat:"6 10E"},
    area:{total:2586, land:2586}},
  people:{population:425017,
    ethnicGroup:"Celtic",
    ethnicGroup:"Portuguese",
    ethnicGroup:"Italian"},
  government:{executive:{chiefOfState:{name:"Jean",...}}}}
```

UnCAL to process data. For example, a function that retrieves all ethnic groups in the graph can be defined simply by

$$\begin{aligned} \text{sfun } f1(\text{ethnicGroup}:t) &= \text{result}:t \\ f1(\ell:t) &= f1(t) \quad \text{for } \ell \neq \text{ethnicGroup} \end{aligned}$$

The keyword **sfun** denotes a function definition by *structural recursion on graphs*. Executing it, we certainly extract:

```
f1(sd) ~> {result:"Celtic", result:"Portuguese", result:"Italian"}
```

The computation by structural recursion on graphs is the *only* computational mechanism of UnCAL. There is no other evaluation mechanism. That situation is analogous to a functional programming language in which the only computation mechanism is “fold”. Although it is restrictive, a benefit of this design choice is that the result of computation by structural recursion always terminates even when an input graph involves cycles. The execution of structural recursion on graphs is different from ordinary evaluation of functional programming. It was realized by two graph algorithms, called the *bulk semantics* and the *recursive semantics*. Although they are called semantics, these are actually algorithms.

The notation $\{t_1, \dots, t_n\}$ is used as an abbreviation of $\{t_1\} \cup \dots \cup \{t_n\}$. UnCAL’s term language consists of markers x , labelled edges $\ell:t$, vertical compositions $s \diamond t$, horizontal compositions $\langle s, t \rangle$, other horizontal compositions $s \cup t$ merging roots, forming

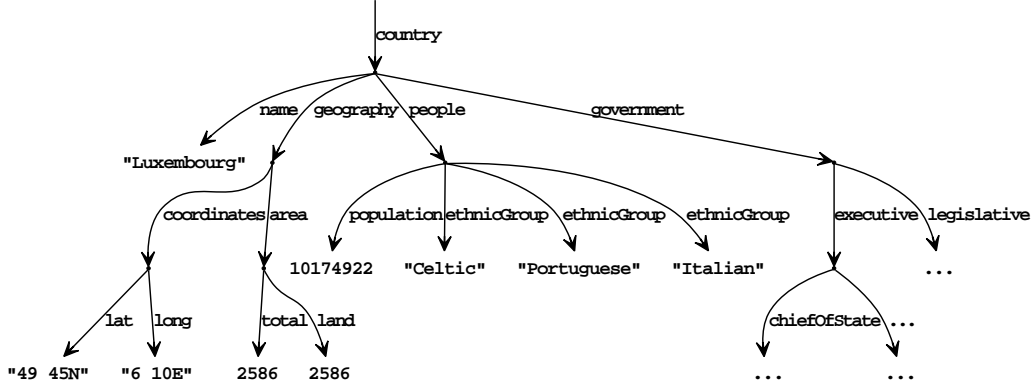


Fig. 1. An example of semi-structured data

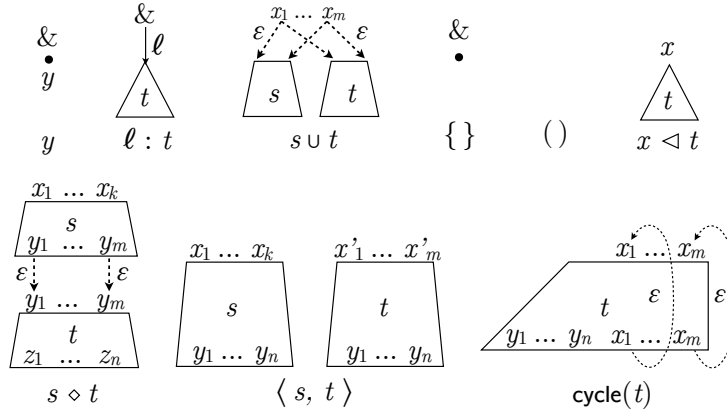


Fig. 2. Graph theoretic definitions of constructors (Buneman et al., 2000)

The notation is slightly changed from the original. The correspondence between the original and this paper's is:

$$\&y = y, \quad @ = \diamond, \quad \oplus = \langle -, - \rangle, \quad (- := -) = (- \triangleleft -).$$

cycles $\text{cycle}(t)$, constants $\{ \}, ()$, and definitions $(x \triangleleft t)$. These term constructions have underlying graph theoretic meaning shown in Fig. 2. Namely, these are officially defined as operations on the ordinary representations of graphs: (vertices set, edges set, roots, leaves)-tuples (V, E, I, O) , but we do not choose this representation as a foundation in this paper. Rather, we reformulate UnCAL in more simpler *algebraic* and *type-theoretic* manner.

UnCAL deals with graphs *modulo bisimulation* (i.e., not modulo graph isomorphism). An UnCAL graph is directed and has (possibly multiple) root(s) written $\&$ (or multiple $x_1 \cdots x_n$) and leaves (written $y_1 \cdots y_m$), and with the roots and leaves drawn pictorially at the top and bottom, respectively. The symbols $x, y_1, y_2, \&$ in the figures and terms are called markers, which are the names of nodes in a graph and are used for references for

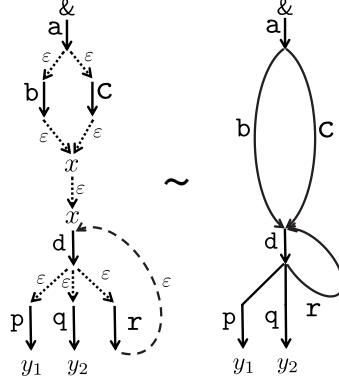


Fig. 3. A graph G and bisimilar one.

cycles and sharings. Here, the edges b and c share the subgraph below the edge d . Also, they are used as port names to connect two graphs. A dotted line labelled ε is called an ε -edge, which is a “virtual” edge connecting two nodes directly. This is achieved by identifying graphs by *extended bisimulation*, which ignores ε -edges suitably in UnCAL. The UnCAL graph G shown at the left in Fig. 3 is an example. This is extended bisimilar to a graph shown at the right in Fig. 3 that reduces all ε -edges. Using UnCAL’s language, G is represented as the following term t_G

$$t_G = a:({b:x} \cup {c:x}) \diamond \text{cycle}(x \triangleleft d:({p:y_1} \cup {q:y_2} \cup {r:x})) .$$

UnCAL’s structural recursive function works also on cyclic data. For example, define another function

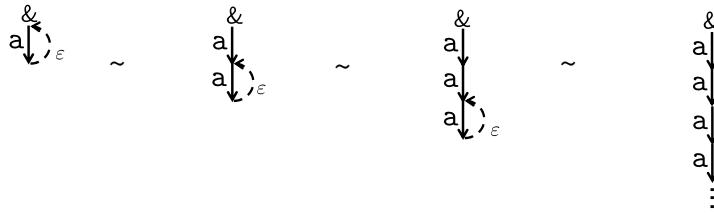
$$\text{sfun } f2(L:T) = a:f2(T)$$

that replaces every edge with a . As expected,

$$f2(t_G) \rightsquigarrow a:({a:x} \cup {a:x}) \diamond \text{cycle}(x \triangleleft a:({a:y_1} \cup {a:y_2} \cup {a:x}))$$

where all labels are changed to a .

Another characteristic role of bisimulation is that it identifies expansion of cycles. For example, a term $\text{cycle}(\& \triangleleft a:\&)$ corresponds to the graph shown below at the leftmost. It is bisimilar to the right ones, especially the infinitely expanded graph shown at the rightmost, which has no cycle.



These are in term notation:

$$\text{cycle}(\& \triangleleft a:\&) \sim a:\text{cycle}(\& \triangleleft a:\&) \sim a:a:\text{cycle}(\& \triangleleft a:\&)$$

1.3.2. The aims of this paper. There have been no algebraic laws that establish the above expansion of `cycle`. Namely, these are merely bisimilar, and not a consequence of any algebraic law. But obviously, we expect that it should be a consequence of the algebraic law of *fixed point property* of `cycle`.

In the original and subsequent formulation of UnCAL (Buneman et al., 2000; Hidaka et al., 2013a, 2010; Asada et al., 2013), there are complications of this kind. The relationship between terms and graphs in UnCAL is not a one-to-one correspondence. There is no explicit term notation for ε -edges. Moreover, ε -edges appear in several different constructions (vertical composition \diamond , merging roots \cup , and `cycle`). No term notation exists for infinite graphs. Hence the rightmost infinite graph of the above expansion of `a` cannot be expressed in syntax. But such an infinite graph is allowed as a possible graph in the original formulation of UnCAL. Consequently, instead of terms, one must use graphs and graph theoretic reasoning with care of bisimulation to reason about UnCAL. Therefore, a property in UnCAL could not be established only by using induction on terms. That fact sometimes makes the proof of a proposition about UnCAL quite complicated. For instance, the proof of the fusion law (Buneman et al., 2000, Theorem 4) is 3.5 pages long (see also our alternative shorter proof in §4.4).

Because UnCAL graphs are identified by bisimulation, it is necessary to use a procedure or algorithm to check the bisimilarity as in the `cycle` example above. Listing some typical valid equations for the bisimulation can be a shortcut (Buneman et al., 2000; Hidaka et al., 2011), but it was only sound and was not complete reasoning method for bisimulation. In this paper, we give a complete equational axiomatisation of UnCAL graphs as an equational logic on the term representation, which captures the original bisimulation.

1.4. Related work

Other than the term representation, various representations of graphs have been known. One of the frequently used representations is by a *system of equations*, also known as an equational term graph (Ariola and Klop, 1996). It is essentially the same as the representation of graphs using **letrec**-expressions used in (Hasegawa, 1997a). Semantically, the least solution of a system of equations is regarded as a (unfolded) graph. Syntactically, a system of (flat) equations can be seen as adjacency lists, e.g. an equation $x = f(y_1, \dots, y_n)$ in a system can be regarded as an adjacency list of vertex x pointed to other vertices y_1, \dots, y_n (cf. discussions in Hamana (2010, Sect. 8)). This representation has also been used as normal forms of process graphs in (Milner, 1984). But the combination of it with structural recursion on graphs has not usually been considered.

One notable exception is a work by Nishimura et al. (1996) on a query language for an object-oriented database. An object in the object-oriented database is represented as a *system of equations*, which may point to other objects, i.e., it forms a *graph*. Although the proposal is done independently of UnCAL, they also developed the “reduce” operation on objects based on a mechanism for data-parallelism on recursive data given in (Nishimura and Ohori, 1999), which is a structural recursion on graphs in our sense (although bisimilar graphs are not identified in their work). This shows that our results would not be restricted to UnCAL, but common in transformations of graph data.

The original formulation of UnCAL focuses a single concrete model, graphs modulo bisimulation. Having this concrete model eases discussions on the complexity of UnCAL transformations. Buneman et al. (2000) showed that UnCAL transformations are in the class FO+TC of problems identified by the first-order logic with transitive closure (Immerman, 1987). The class FO+TC is the same as NL (Immerman, 1987), and thus UnCAL queries are performed in polynomial time in nature. They also showed that, for tree-encoded relational data, UnCAL has the exactly same power as the relational algebra, i.e., first-order formulas, because finite unfolding of transitive closure suffices for such trees.

As an extension of UnCAL to handle graphs other than unordered ones, Hidaka et al. (2013a) proposed an UnCAL variant for ordered graphs, and then Asada et al. (2013) extended the result to more general graphs whose branching structure is specified by a certain sort of monads. Accordingly, they gave new formalization of bisimulation to such graph models.

An earlier version of this paper appeared in (Hamana, 2015), which used iteration algebras for the completeness proof. The present paper refines the mathematical setting throughout and investigates it further in various directions. We use iteration categories as the foundations instead of iteration algebras, which provide a unified framework for graphs, bisimulation, and structural recursion.

1.5. Organisation

This paper is organised as follows. We first give an equational theory for UnCAL graphs by reformulating UnCAL graph data in a type theoretic manner in Section 2. We then give categorical semantics of UnCAL using iteration categories in Section 3.1. We further model the computation mechanism “structural recursion on graphs” in Section 4. We prove completeness of our axioms for UnCAL graphs for bisimulation in Section 5. We consider several instances of categorical models in Section 6. In particular, in Section 6.3, we present the λ G-calculus as a model, and show that it induces a sound and complete translation from UnCAL to the λ G-calculus. In Section 7, we discuss variations of UnCAL and related graph calculi.

2. UnCAL and its Equational Theory

In this section, we establish a new framework of equational reasoning for UnCAL graphs without explicitly touching graphs. Namely, we *reformulate* UnCAL as an equational logic in a type theoretic manner. We do not employ the graph-theoretic or operational concepts (such as ε -edges, bisimulation, and the graph theoretic definitions in Fig. 2). Instead, we give a *syntactic* equational axiomatisation of UnCAL graphs following the tradition of categorical type theory (e.g. (Crole, 1993)).

The idea of our formulation is as follows. The general methodology of categorical type theory is to interpret a term-in-context $\Gamma \vdash t : \tau$ in a type theory as a morphism $\llbracket t \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \tau \rrbracket$ in a category. Drawing the morphism $\llbracket t \rrbracket$ as a tree diagram in which the

top is the root for $\llbracket \tau \rrbracket$, it is a graph consisting directed edges that go from the bottom $\llbracket \Gamma \rrbracket$ to the top $\llbracket \tau \rrbracket$. We aim to model UnCAL graphs in this way.

The syntax of UnCAL in this paper is slightly modified from the original presentation (cf. Fig. 2) to reflect the categorical idea, and to make UnCAL more readable for the reader familiar with categorical type theory, but this is essentially just a notational change. This replacement is one-to-one, and one can systematically recover the original syntax from the syntax used here.

2.1. Syntax

2.1.1. Markers and contexts. We assume an infinite set of symbols called *markers*, denoted by typically x, y, z, \dots . One can understand markers as variables in a type theory. The marker named $\&$ is called the default marker (cf. Remark 2.1). Let L be a set of labels. A *label* ℓ is a symbol (e.g. $\mathbf{a}, \mathbf{b}, \mathbf{c}, \dots$ in Fig. 3). A *context*, denoted by $\langle\langle x_1, x_2, \dots \rangle\rangle$, is a finite sequence of pairwise distinct markers. We typically use X, Y, Z, \dots for contexts. We use $\langle\langle \rangle\rangle$ for the empty contexts, $X + Y$ or X, Y for the concatenation, and $|X|$ for its length. We may use the vector notation \vec{x} for sequence x_1, \dots, x_n . The outermost bracket $\langle\langle \rangle\rangle$ of a context may be omitted. We may use an alternative notation for the empty context: $0 = \langle\langle \rangle\rangle$. Note that the concatenation may need suitable renaming to satisfy pairwise distinctness of markers.

2.1.2. Raw terms.

$$t ::= y_Y \mid \ell : t \mid s \diamond t \mid \langle s, t \rangle \mid \text{cycle}^X(t) \mid \{\}_Y \mid ()_Y \mid \lambda_Y \mid (x \triangleleft t)$$

We assume several conventions to simplify the presentation of theory. We often omit subscripts or superscripts such as Y when they are unimportant or inferable. We identify $\langle t, () \rangle$ and $\langle (), t \rangle$ with t , and $\langle \langle s, t \rangle, u \rangle$ with $\langle s, \langle t, u \rangle \rangle$; thus we will freely omit parentheses as $\langle t_1, \dots, t_n \rangle$. When writing $\langle t_1, \dots, t_n \rangle$ for any $n \in \mathbb{N}$, we mean that it includes the cases $n = 0, 1$. If $n = 0$, it means $()$. If $n = 1$, it means t_1 (not a tuple).

A constant λ is used to express a branch in a tree, and we call the symbol λ a *man*, because it is similar to the shape of a kanji or Chinese character meaning a man. A definition $(x \triangleleft t)$ does not bind x . As a graph, it is intended to name the root as x . See Fig. 2 and the typing rule (Def) in (Fig. 4).

2.2. Typed syntax

For contexts X, Y , we inductively define a judgment relation

$$Y \vdash t : X$$

of terms by the typing rules in Fig. 4. Now Y is similar to a variable context in ordinary type theories, which we call the *source context* and X is the names of roots, which we call the *target context* or *type*. We identify t of type $\&$ with $(\& \triangleleft t)$.

The intuitive meaning of the term constructors (i.e. \diamond , $-:-$, etc.) is the corresponding operations on graphs described in Fig. 2. But we warn that in our formulation, Fig. 2

$$\begin{array}{c}
\text{(Nil)} \frac{}{Y \vdash \{\}_Y : \&} \quad \text{(Emp)} \frac{}{Y \vdash ()_Y : \langle\!\langle\!\rangle\!\rangle} \quad \text{(Man)} \frac{}{y_1, y_2 \vdash \wedge_{\langle\!\langle y_1, y_2 \rangle\!\rangle} : \&} \\
\\
\text{(Com)} \frac{Y \vdash s : Z \quad X \vdash t : Y}{X \vdash s \diamond t : Z} \quad \text{(Label)} \frac{\ell \in L \quad Y \vdash t : \&}{Y \vdash \ell : t : \&} \quad \text{(Mark)} \frac{Y = \langle\!\langle y_1, \dots, y_n \rangle\!\rangle}{Y \vdash y_{i_Y} : \&} \\
\\
\text{(Pair)} \frac{Y \vdash s : X_1 \quad Y \vdash t : X_2}{Y \vdash \langle s, t \rangle : X_1 + X_2} \quad \text{(Cyc)} \frac{Y + X \vdash t : X}{Y \vdash \text{cycle}^X(t) : X} \quad \text{(Def)} \frac{Y \vdash t : \&}{Y \vdash (x \triangleleft t) : x}
\end{array}$$

Fig. 4. Typing rules

is not the definitions of constructors, so the reader should use it just for intuition. If the reader is familiar with category theory, it may be helpful to refer the categorical interpretation we will define in Fig. 10 (but not mandatory). After establishing our categorical semantics, we can give a graph theoretic model (Appendix A).

For example, the term $\mathbf{t_G}$ in §1 is well-typed as

$$y_1, y_2 \vdash \mathbf{t_G} : \&,$$

which corresponds a graph in Fig. 3. When a raw term t is well-typed by the typing rules, we call t a (well-typed UnCAL) term.

Remark 2.1. In UnCAL, we always need to name the root of a term by a marker. By default, we name the root as “ $\&$ ” as the above $\mathbf{t_G}$, hence we call $\&$ the *default marker*. This convention is reflected to the typing rules. \square

2.2.1. On (Pair). We have assumed in §2.1.1 that if contexts X_1 and X_2 share some markers, the concatenation $X_1 + X_2$ need renaming of markers to satisfy pairwise disjointness. We do not explicitly define how to rename markers to avoid clutter. This also gives a simplified notation for pair terms.

For example, suppose that $\langle\!\langle\&\!\rangle\!\rangle + \langle\!\langle\&\!\rangle\!\rangle$ is renamed to y_1, y_2 . Consider (Pair) construct

$$\text{(Pair)} \frac{Y \vdash t_1 : \& \quad Y \vdash t_2 : \&}{Y \vdash \langle t_1, t_2 \rangle : \langle\!\langle\&\!\rangle\!\rangle + \langle\!\langle\&\!\rangle\!\rangle}$$

Hence $Y \vdash \langle t_1, t_2 \rangle : y_1, y_2$. We can derive essentially the same term

$$\text{(Pair)} \frac{Y \vdash (y_1 \triangleleft t_1) : y_1 \quad Y \vdash (y_2 \triangleleft t_2) : y_2}{Y \vdash \langle (y_1 \triangleleft t_1), (y_2 \triangleleft t_2) \rangle : y_1, y_2}$$

without renaming markers in the target context. We identify these two term judgements and we often choose the first representation $Y \vdash \langle t_1, t_2 \rangle : y_1, y_2$, because it is simpler and the part of definitions $y_1 \triangleleft -$ and $y_2 \triangleleft -$ can be recovered from the target context y_1, y_2 . Hence hereafter, writing simply $X \vdash \langle t_1, \dots, t_n \rangle : y_1, \dots, y_n$, we mean $X \vdash \langle (y_1 \triangleleft t_1), \dots, (y_n \triangleleft t_n) \rangle : y_1, \dots, y_n$. As long as well-typed terms, there is no confusion.

Even if the markers in the target context are omitted, still the information for this renaming can be recovered from contexts around the term. For example, when we consider

a well-typed term $s \diamond \langle t_1, t_2 \rangle$ where the source context of s is y_1, y_2 , then we can rename the term correctly as $s \diamond \langle (y_1 \triangleleft t_1), (y_2 \triangleleft t_2) \rangle$.

Remark 2.2. We could omit the marker in a target context and also the syntax $(x \triangleleft -)$ and then, we could replace, e.g., (Cyc) with:

$$\frac{Y + X \vdash t : n \quad |X| = n}{Y \vdash \text{cycle}^X(t) : n}$$

But in this paper we keep the UnCAL-style syntax to accommodate with the existing work on UnCAL. \square

2.2.2. Abbreviations. We define several abbreviations for terms. We define

$$\pi_{X,Y}^X \triangleq \langle x_1, \dots, x_n \rangle$$

for $X = \langle x_1, \dots, x_n \rangle$. When $n = 1$, $\pi_{x_1,Y}^{x_1} \triangleq x_1$. When $n = 0$, $\pi_{0,Y}^0 \triangleq ()$. Similarly for the case $\pi_{X,Y}^Y$, but because of the convention of $\langle -, - \rangle$ describe in §2.2.1, it is better to explicitly mention the source and target contexts when defining an abbreviation. So we use the following style

$$\frac{X = \langle x_1, \dots, x_n \rangle}{X + Y \vdash \pi_{X,Y}^X \triangleq \langle x_1, \dots, x_n \rangle : X} \quad \frac{Y = \langle y_1, \dots, y_n \rangle}{X + Y \vdash \pi_{X,Y}^Y \triangleq \langle y_1, \dots, y_n \rangle : Y}$$

These mean to define the abbreviations and to indicate also the source and target contexts of the abbreviations.

$$\begin{aligned} & \frac{Y_1 \vdash t_1 : X_1 \quad Y_2 \vdash t_2 : X_2}{Y_1 + Y_2 \vdash t_1 \times t_2 \triangleq \langle t_1 \diamond \pi_{Y_1,Y_2}^{Y_1}, t_2 \diamond \pi_{Y_1,Y_2}^{Y_2} \rangle : X_1 + X_2} \\ & \frac{Y_1 \vdash t_1 : \& \quad Y_2 \vdash t_2 : \&}{Y_1 + Y_2 \vdash \{t_1\} \cup \{t_2\} \triangleq \& \diamond \langle t_1, t_2 \rangle : \&} \\ & \frac{}{0 \vdash \text{id}_{\langle \rangle} \triangleq ()_{\langle \rangle} : 0} \quad \frac{}{x \vdash \text{id}_{\langle x \rangle} \triangleq (x \triangleleft x) : x} \quad \frac{X = \langle x_1, \dots, x_n \rangle}{X \vdash \text{id}_X \triangleq \langle x_1, \dots, x_n \rangle : X} \\ & \frac{X \vdash \text{id}_X : X}{X \vdash \Delta_X \triangleq \langle \text{id}_X, \text{id}_X \rangle : X + X} \quad \frac{x, y \vdash \pi_{x,y}^x : x \quad x, y \vdash \pi_{x,y}^y : y}{x, y \vdash c_{x,y} \triangleq \langle \pi_{x,y}^y, \pi_{x,y}^x \rangle : y, x} \end{aligned}$$

Inheriting the convention of $\langle -, - \rangle$, we also identify $(s \times t) \times u$ with $s \times (t \times u)$, thus we omit parentheses as $t_1 \times \dots \times t_n$. We will see that these named terms are actually intended arrows in a cartesian category (§3.1).

Definition 2.3. (Substitution) Suppose

$$y_1, \dots, y_n \vdash t : X, \quad Z \vdash s_i : \& \quad (1 \leq i \leq n).$$

$$\begin{array}{c}
\text{(Ax)} \frac{(Y \vdash s = t : X) \in E}{Y \vdash s = t : X} \quad \text{(Sub)} \frac{y_1, \dots, y_n \vdash t = t' : X \quad Z \vdash s_i = s'_i : \& \quad (1 \leq i \leq n)}{Z \vdash t [\vec{y} \mapsto \vec{s}] = t' [\vec{y} \mapsto \vec{s}'] : X} \\
\\
\text{(Com)} \frac{Y \vdash s = s' : Z \quad Y \vdash t = t' : X}{X \vdash s \diamond t = s' \diamond t' : Z} \quad \text{(Pair)} \frac{Y \vdash s = s' : X_1 \quad Y \vdash t = t' : X_2}{Y \vdash \langle s, t \rangle = \langle s', t' \rangle : X_1 + X_2} \\
\\
\text{(Cyc)} \frac{Y + X \vdash t = t' : X}{Y \vdash \text{cycle}^X(t) = \text{cycle}^{X'}(t) : X} \quad \text{(Def)} \frac{Y \vdash t = t' : \&}{Y \vdash (x \triangleleft t) = (x \triangleleft t') : x} \\
\\
\text{(Ref)} \frac{}{Y \vdash t = t : X} \quad \text{(Sym)} \frac{Y \vdash s = t : X}{Y \vdash t = s : X} \quad \text{(Tr)} \frac{Y \vdash s = t : X \quad Y \vdash t = u : X}{Y \vdash s = u : X}
\end{array}$$

Fig. 5. Equational Logic EL-UnCAL

Then a substitution $Z \vdash t [\vec{y} \mapsto \vec{s}] : X$ is inductively defined as follows.

$$\begin{array}{ll}
y_i [\vec{y} \mapsto \vec{s}] \triangleq s_i & (t_1 \diamond t_2) [\vec{y} \mapsto \vec{s}] \triangleq t_1 \diamond (t_2 [\vec{y} \mapsto \vec{s}]) \\
\{\}_Y [\vec{y} \mapsto \vec{s}] \triangleq \{\}_Z & \langle t_1, t_2 \rangle [\vec{y} \mapsto \vec{s}] \triangleq \langle t_1 [\vec{y} \mapsto \vec{s}], t_2 [\vec{y} \mapsto \vec{s}] \rangle \\
(\)_Y [\vec{y} \mapsto \vec{s}] \triangleq (\)_Z & \text{cycle}^X(t) [\vec{y} \mapsto \vec{s}] \triangleq \text{cycle}^X(t [\vec{y} \mapsto \vec{s}], \vec{x} \mapsto \vec{x}) \\
(\ell : t) [\vec{y} \mapsto \vec{s}] \triangleq \ell : (t [\vec{y} \mapsto \vec{s}]) & (x \triangleleft t) [\vec{y} \mapsto \vec{s}] \triangleq (x \triangleleft t [\vec{y} \mapsto \vec{s}]) \\
\wedge \langle y_1, y_2 \rangle [y_1 \mapsto s_1, y_2 \mapsto s_2] \triangleq \wedge \langle y_1, y_2 \rangle \diamond \langle s_1, s_2 \rangle
\end{array}$$

Note that $t [\vec{y} \mapsto \vec{s}]$ denotes a meta-level substitution operation, not an explicit substitution. This simultaneous substitution is a generalisation of the single variable case given in (Hidaka et al., 2011), where the contexts and types for terms were not explicit. Explicating the context and type information is important in our development to connect it with the categorical interpretation (cf. Lemma 3.4).

2.3. Equational axiomatisation

For terms $Y \vdash s : X$ and $Y \vdash t : X$, an (*UnCAL*) *equation* is of the form

$$Y \vdash s = t : X.$$

Hereafter, we often omit the source X and target Y contexts, and simply write $s = t$ for an equation, but even such an abbreviated form, we assume that it has implicitly suitable source and target contexts and is of the above judgemental form.

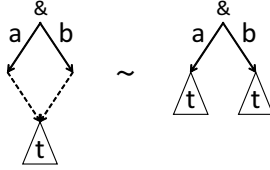
A set of *UnCAL axioms* E is given by the axioms AxG in Fig. 6 plus a set of arbitrary equations called *additional axioms*. The axioms AxG characterises UnCAL graphs. These axioms are chosen to soundly and completely represent the original bisimulation of graphs by the equality of this logic. Checking soundness is straightforward: for every axiom $s = t$, we see that s and t are bisimilar. But completeness is not clear only from the axioms. We will show it in §5.

The equational logic EL-UnCAL for UnCAL is a logic to deduce formally proved equations, called (*UnCAL*) *theorems*. The equational logic is almost the same as ordinary one for algebraic terms. The inference system of equational logic for UnCAL terms is given in

Fig. 5. The structural rules (weakening, contraction, and permutation of source context Y) are derivable from (Sub). The set of all theorems deduced from axioms E is called a (*UnCAL*) *theory*. When E has no additional axioms, we call the theory *pure*.

2.3.1. Meaning of axioms. The Fig. 7 and Fig. 8 show how each axiom in \mathbf{AxG} can be understood graphically and intuitively by regarding UnCAL terms as graphs. Not only giving us intuition, this graphical interpretation can be stated formally. In Appendix A, we will show that these graphs are obtained as the interpretations of terms using a categorical model of \mathbf{AxG} in a particular category for UnCAL, i.e., the category \mathbf{Gr} of UnCAL graphs.

We explain the meaning of the axioms in \mathbf{AxG} in further detail. The axiom (sub) is similar to the β -reduction in the λ -calculus. With the axiom (SP), it induces the axioms for cartesian product (Lambek and Scott, 1986) (cf. the **derived theory** in §2.3.3). It induces also a characteristic property of the UnCAL graphs, namely, shared graphs are copyable. For example, the following two graphs are bisimilar (thus identified) in the original formulation.



In our equational theory, it is a theorem by (sub):

$$\wedge \diamond (\langle \mathbf{a} : \&, \mathbf{b} : \& \rangle \diamond t) = \wedge \diamond \langle \mathbf{a} : t, \mathbf{b} : t \rangle.$$

Thus UnCAL graphs should not be interpreted in a mere monoidal category, unlike other monoidal categorical formulations of graphs or DAGs, such as (Gibbons, 1995; Hasegawa, 1997a; Corradini and Gadducci, 1999; Fiore and Campos, 2013), where shared graphs are not copyable. The cartesian structure also provides a canonical commutative comonoid with comultiplication Δ .

Two terms are paired with a common root by $\{s\} \cup \{t\} = \wedge \diamond \langle s, t \rangle$. The commutative monoid axioms states that this pairing $\{-\} \cup \{-\}$ can be parenthesis-free in the nested case. The degenerate bialgebra axioms state the compatibility between the commutative monoid and comonoid structures. The degenerated bialgebra is suitable to model directed acyclic graphs (cf. (Fiore and Campos, 2013, §4.5)), where it is stated for a strict monoidal category (known as a PROP) (Mac Lane, 1965). The monoid multiplication \wedge expresses a branch in a tree, while the comultiplication Δ expresses a sharing. Commutativity expresses that there is no order between the branches of a node, cf. (commu \cup) in the **derived theory**.

The axiom (c1) and (c2) delete trivial cycles as graphs (see Fig. 7 in Appendix). Similar axioms can be found in other axiomatisations of bisimulation, e.g. (Milner, 1984; Bloom et al., 1993; Ésik, 2000).

Parameterised fixed-point axioms axiomatise cycle as a fixed point operator. They (minus (CI)) are equivalent to the axioms for Conway operators of Bloom and Ésik

Composition

$$\begin{array}{ll}
\text{(sub)} & Z \vdash t \diamond \langle s_1, \dots, s_n \rangle = t [\vec{y} \mapsto \vec{s}] : X \\
& \text{for } Y \vdash t : X \quad Y = \langle\langle y_1, \dots, y_n \rangle\rangle \\
& \quad Z \vdash s_1 : \& \dots Z \vdash s_n : \& \\
\text{(SP)} & Z \vdash \langle \pi_{X,Y}^X \diamond t, \pi_{X,Y}^Y \diamond t \rangle = t : X + Y \\
& \text{for } Z \vdash t : X + Y \\
(\eta \text{ } \wedge) & x, y \vdash \wedge \diamond \langle x, y \rangle = \wedge_{\langle\langle x, y \rangle\rangle} : \&
\end{array}$$

Parameterised fixed point

$$\begin{array}{ll}
\text{(fix)} & Y \vdash \text{cycle}(t) = t \diamond \langle \text{id}_Y, \text{cycle}(t) \rangle : X \\
& \text{for } Y + X \vdash t : X \\
\text{(nat)} & Z \vdash \text{cycle}(t) \diamond s = \text{cycle}(t \diamond (s \times \text{id}_X)) : X \\
& \text{for } Y + X \vdash t : X \quad Z \vdash s : Y \\
\text{(dinat)} & Y \vdash \text{cycle}(s \diamond t) = s \diamond \text{cycle}(t \diamond (\text{id}_Y \times s)) : X \\
& \text{for } Z \vdash s : X \quad Y + X \vdash t : Z \\
\text{(Bekić)} & Z \vdash \text{cycle}^{X+Y}(\langle t, s \rangle) = \langle \pi_{Z,X}^X, \text{cycle}^Y(s) \rangle \diamond \\
& \quad \langle \text{id}_Z, \text{cycle}^X(t \diamond \langle \text{id}_{Z+X}, \text{cycle}^Y(s) \rangle) \rangle : X + Y \\
& \text{for } Z + X + Y \vdash t : X \quad Z + X + Y \vdash s : Y \\
\text{(CI)} & X \vdash \text{cycle}(\langle \begin{array}{c} t \diamond (\text{id}_X \times \rho_1), \\ \dots, \\ t \diamond (\text{id}_X \times \rho_m) \end{array} \rangle) = \Delta_m \diamond \text{cycle}(t \diamond (\text{id}_X \times \Delta_m)) : Y \\
& \text{for } X + Y \vdash t : \&
\end{array}$$

Deleting trivial cycle

$$\begin{array}{ll}
\text{(c1)} & \vdash \text{cycle}^{\&}(\&) = \{\} : \& \\
\star \text{(c2)} & y \vdash \text{cycle}^{\&}(\wedge_{\langle\langle y, \& \rangle\rangle}) = y : \&
\end{array}$$

Degenerated commutative bialgebra

$$\begin{array}{ll}
\text{(unitL } \wedge) & x \vdash \wedge_{\langle\langle \&, x \rangle\rangle} \diamond \{\}_0 \times \text{id}_x = x : \& \\
\text{(unitR } \wedge) & x \vdash \wedge_{\langle\langle x, \& \rangle\rangle} \diamond (\text{id}_x \times \{\}_0) = x : \& \\
\text{(assoc } \wedge) & x, y, z \vdash \wedge \diamond (\text{id} \times \wedge) = \wedge \diamond (\wedge \times \text{id}) : \& \\
\star \text{(com } \wedge) & x, y \vdash \wedge \diamond \mathbf{c}_{x,y} = \wedge_{\langle\langle x, y \rangle\rangle} : \& \\
\star \text{(degen)} & x \vdash \wedge \diamond \langle x, x \rangle = x : \&
\end{array}$$

In (sub), n is possibly 0 or 1. If $n = 0$, the left-hand side is $t \diamond ()$. If $n = 1$, the left-hand side is $t \diamond s_1$. In (CI), for $Y = \langle\langle y_1, \dots, y_m \rangle\rangle$, $\& \vdash \Delta_m \triangleq \langle \text{id}_{\&}, \dots, \text{id}_{\&} \rangle : Y$, $Y \vdash \rho_i \triangleq \langle y_{i_1}, \dots, y_{i_m} \rangle : Y$, where $i_1, \dots, i_m \in \{1, \dots, m\}$, meaning that some of i_1, \dots, i_m may be equal.

Fig. 6. Axioms AxG

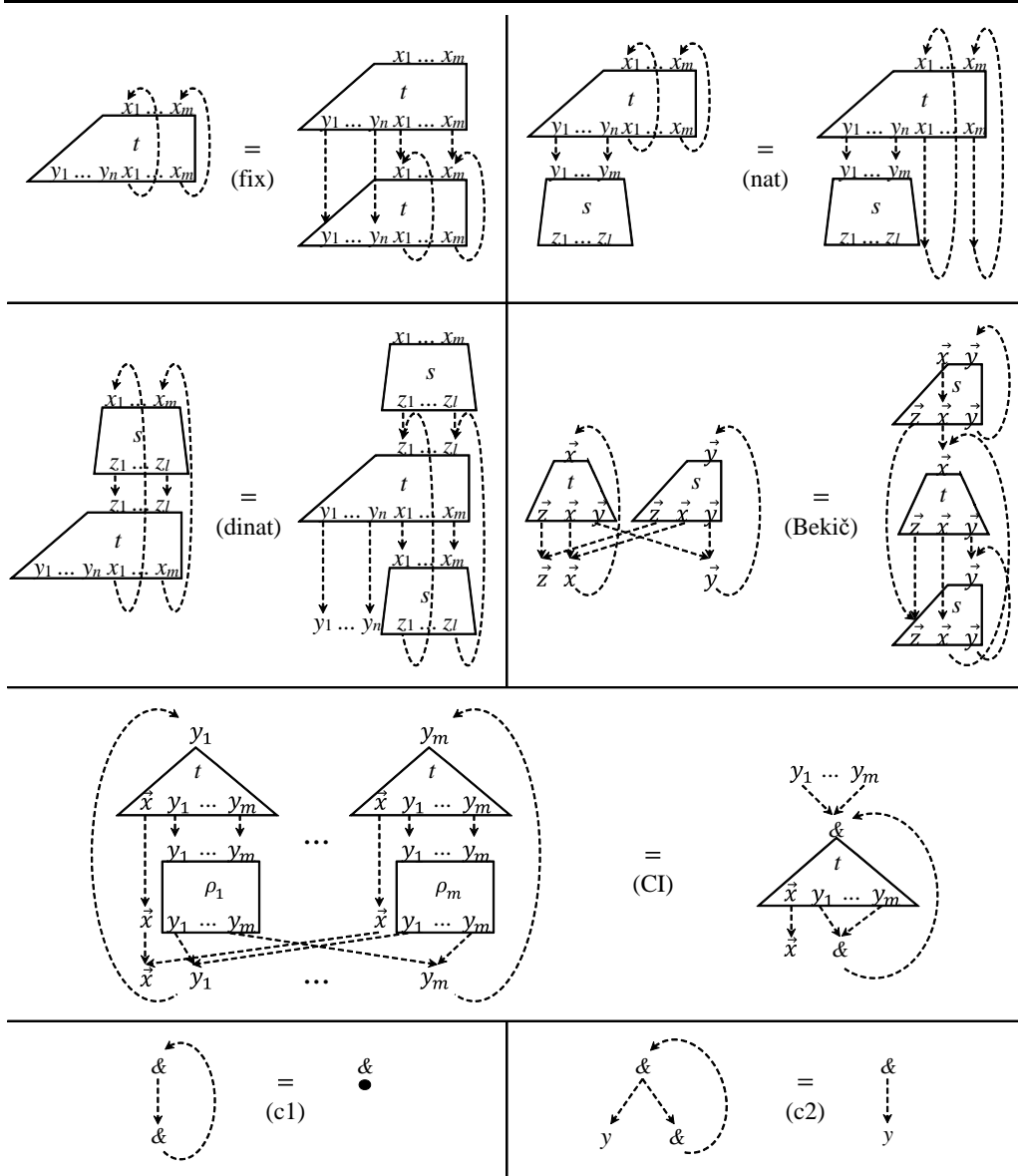


Fig. 7. Graphical representation of the axioms AxG

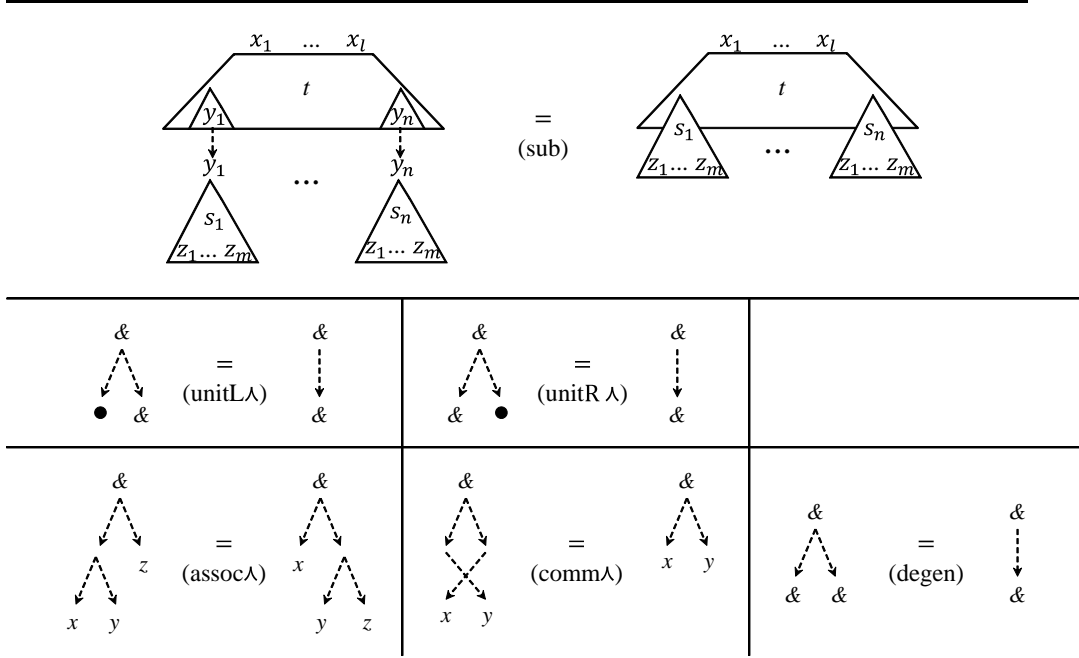


Fig. 8. Graphical representation of the axioms AxG (continued)

(tmnl)	t	$=$	$(\)_Y$ for all $Y \vdash t : \langle\!\langle\!\rangle\!\rangle$
(fst)	$\pi_{X,Y}^X \diamond \langle s, t \rangle$	$=$	s
(snd)	$\pi_{X,Y}^Y \diamond \langle s, t \rangle$	$=$	t
(dpair)	$\langle t_1, t_2 \rangle \diamond s$	$=$	$\langle t_1 \diamond s, t_2 \diamond s \rangle$
(fsi)	$\langle \pi_{X,Y}^X, \pi_{X,Y}^Y \rangle$	$=$	$\text{id}_{X,Y}$
(bmul)	$(\)_{\&} \times (\)_{\&}$	$=$	$(\)_{\&} \diamond \wedge$
(bcomul)	$\Delta \diamond \{ \}_0$	$=$	$\{ \}_0 \times \{ \}_0$
(unR \diamond)	$t \diamond \text{id}$	$=$	t
(unL \diamond)	$\text{id} \diamond t$	$=$	t
(assoc \diamond)	$(s \diamond t) \diamond u$	$=$	$s \diamond (t \diamond u)$
(bunit)	$(\)_{\&} \diamond \{ \}_0$	$=$	id_0
(compa)	$\Delta \diamond \wedge$	$=$	$(\wedge \times \wedge) \diamond (\text{id} \times \text{c} \times \text{id}) \diamond (\Delta \times \Delta)$
(comm \cup)	$\{s\} \cup \{t\}$	$=$	$\{t\} \cup \{s\}$
(unit \cup)	$\{ \} \cup \{t\}$	$=$	$t = \{t\} \cup \{ \}$
(assoc \cup)	$\{ \{s\} \cup \{t\} \} \cup \{u\}$	$=$	$\{s\} \cup \{ \{t\} \cup \{u\} \}$
(degen')	$\{t\} \cup \{t\}$	$=$	t

Fig. 9. Derived theory

(1993). Bekič law is well-known in denotational semantics (cf. (Winskel, 1993, §10.1)), which says that the fixed point of a pair can be obtained by computing the fixed point of each of its components independently and composing them suitably (see also Fig. 7 in Appendix). Conway operators have also arisen in work independently of Hyland and Hasegawa (1997a), who established a connection with the notion of traced cartesian categories (Joyal et al., 1996).

2.3.2. On commutative identities. There are equalities that holds in the cpo semantics but Conway operators do not satisfy; e.g. $\text{cycle}(t) = \text{cycle}(t \diamond t)$ does not hold by the Conway axioms. The axiom (CI) fills this gap. It corresponds to the commutative identities of Bloom and Ésik (1993), which ensures that all equalities that hold in the cpo semantics do hold. See also (Simpson and Plotkin, 2000, Section 2) for a useful overview around this. This form is taken from it and adopted to the UnCAL setting.

2.3.3. Derived theory. The law (sub) is useful to simplify terms. The law has not been formulated in UnCAL literature, because no simultaneous substitution has been formulated. It will also be used to compute normal forms of UnCAL terms to prove completeness for bisimulation in §5. The Fig. 9 shows some UnCAL theorems, which are formally derivable from the axioms, or are proved using induction on typing derivations. We write $\{\} \cup \{t\}$ for $\{\{\}\} \cup \{t\}$.

The meaning of these theorems are as follows.

- (i) The first three lines with (SP) in AxG indicate that UnCAL has the cartesian product (cf. (Lambek and Scott, 1986; Amadio and Curien, 1998)). (fst), (snd), (fsi) and (dpair) are proved by expanding the abbreviations and (sub),(SP) in EL-UnCAL. (tmnl) is proved by induction on typing derivations and EL-UnCAL.
- (ii) (bmul), (bcomul), (bunit) and (compa) are a part of the axioms of commutative bialgebra. In our setting these are theorems because it has the cartesian product, see Remark 3.3.
- (iii) (unR \diamond), (unL \diamond) and (assoc \diamond) state that \diamond is the composition of morphisms. These are proved by (sub).
- (iv) The rest of theorems (comm \cup)-(degen') states that \cup has the unit $\{\}$, and is associative, commutative, and idempotent. These are derivable from the axioms of **Degenerated commutative bialgebra** through the abbreviation

$$\{s\} \cup \{t\} = \wedge \diamond \langle s, t \rangle.$$

These theorems for \cup are very important in UnCAL. They validate that the notation $\{s\} \cup \{t\}$ can be understood as the set union operation. As a graph, the constructor \cup represents a commutative and idempotent branch.

Since our modelling is categorical, we did not directly choose these (comm \cup)-(degen') as axioms. Rather, we chose the categorical monoid laws (Mac Lane, 1971) in a cartesian category as the axioms (**Degenerated commutative monoid**) with the

unit $\{\}$ and multiplication \wedge . See the proof of Prop. 3.8 for details of the categorical modelling.

The axioms AxG include a few redundant axioms, but they are useful for understanding. Actually, the axioms (c1) and (unitR \wedge) are derivable. The axiom (unitR \wedge) is obtained from (unitL \wedge) and (comm \wedge). For (c1), the proof is

$$\text{cycle}(\&) =^{(\text{unitL}\wedge)} \text{cycle}(\wedge \diamond (\{\}_0 \times \text{id})) =^{(\text{nat})} \text{cycle}^\&(\wedge \langle\langle y, \& \rangle\rangle) \diamond \{\}_0 =^{(\text{c2})} y \diamond \{\}_0 = \{\}_0.$$

3. Categorical Semantics

In this section, we give categorical semantics of UnCAL graphs, and show its categorical completeness.

We interpret edges of an UnCAL graph as *morphisms* (of the opposite directions), the vertical composition \diamond as the *composition of morphisms*, and *cycle* as a *fixed point operator* in a suitable category. Thus the target categorical structure should have a notion of fixed point, which has been studied in iteration theories of Bloom and Ésik (1993). In particular, iteration categories (Ésik, 1999a) are suitable, which are traced cartesian categories (Joyal et al., 1996) additionally satisfying the commutative identities axiom (Bloom and Ésik, 1993).

We write $\mathbf{1}$ for the terminal object, \times for the cartesian product, π_1, π_2 for the first and second projections, $\langle -, - \rangle$ for pairing, and $\Delta = \langle \text{id}, \text{id} \rangle$ for diagonal in a cartesian category.

Definition 3.1. (Ésik, 1999a; Bloom and Ésik, 1993) A *Conway operator* in a cartesian category \mathcal{C} is a family of functions $(-)^{\dagger} : \mathcal{C}(A \times X, X) \rightarrow \mathcal{C}(A, X)$ satisfying:

$$\begin{aligned} (f \circ (g \times \text{id}_X))^{\dagger} &= f^{\dagger} \circ g \\ (f^{\dagger})^{\dagger} &= (f \circ (\text{id}_A \times \Delta))^{\dagger} \\ f \circ \langle \text{id}_A, (g \circ \langle \pi_1, f \rangle)^{\dagger} \rangle &= (f \circ \langle \pi_1, g \rangle)^{\dagger}. \end{aligned}$$

An *iteration category* is a cartesian category having a Conway operator additionally satisfying the “commutative identities”

$$\langle f \circ (\text{id}_X \times \rho_1), \dots, f \circ (\text{id}_X \times \rho_m) \rangle^{\dagger} = \Delta_m \circ (f \circ (\text{id}_X \times \Delta_m))^{\dagger} : X \rightarrow A^m$$

where

- $f : X \times A^m \rightarrow A$
- diagonal $\Delta_m \triangleq \langle \text{id}_A, \dots, \text{id}_A \rangle : A \rightarrow A^m$
- $\rho_i : A^m \rightarrow A^m$ such that $\rho_i = \langle q_{i1}, \dots, q_{im} \rangle$ where each q_{ij} is one of projections $\pi_1, \dots, \pi_m : X^m \rightarrow X$.

An *iteration functor* between iteration categories is a cartesian functor that preserves Conway operators.

We may denote by $(\mathcal{C}, (-)^{\dagger})$ an iteration category, and take it as the basic structure for the interpretation of an UnCAL theory. A typical example of iteration category is

the category of cpos and continuous functions (Bloom and Ésik, 1993; Hasegawa, 1997a), where the least fixed point operator is a Conway operator, see §6.2.

Definition 3.2. Given a set L , an L -monoid $(A, \llbracket - \rrbracket_L^A)$ in an iteration category $(\mathcal{C}, (-)^\dagger)$ consists of

- (i) a commutative monoid object $(A, \eta^A : \mathbf{1} \rightarrow A, \mu^A : A \times A \rightarrow A)$ in \mathcal{C} satisfying

$$(\mu^A)^\dagger = \text{id}_A,$$

- (ii) a function $\llbracket - \rrbracket_L^A : L \rightarrow \mathcal{C}(A, A)$ that assigns to each $\ell \in L$, a morphism $\llbracket \ell \rrbracket_L^A : A \rightarrow A$ in \mathcal{C} .

Let A in \mathcal{C} and B in \mathcal{D} be L -monoids. We say that an iteration functor $F : \mathcal{C} \rightarrow \mathcal{D}$ *preserves L -monoid structures* if $F(A) = B$, F preserves monoid structures, and $F(\llbracket \ell \rrbracket_L^A) = \llbracket \ell \rrbracket_L^B$ for every $\ell \in L$.

Remark 3.3. An L -monoid A is not merely a monoid, but also automatically a *degenerated commutative bialgebra* (cf. (Fiore and Campos, 2013)) in a cartesian category \mathcal{C} . Namely A is also a comonoid $(A, !, \Delta)$ that satisfies the compatibility

$$\Delta \circ \eta^A = \eta^A \times \eta^A, \quad \Delta \circ \mu^A = (\mu^A \times \mu^A) \circ (\text{id} \times c \times \text{id}) \circ (\Delta \times \Delta), \quad \mu^A \circ \Delta = \text{id}$$

where $c = \langle \pi_2, \pi_1 \rangle$. The last equation is by $\mu^A \circ \Delta = \mu^A \circ \langle \text{id}, \text{id} \rangle = \mu^A \circ \langle \text{id}, (\mu^A)^\dagger \rangle = (\text{fix}) (\mu^A)^\dagger = \text{id}$. Thus, L -monoid structure is suitable to model branch and sharing in UnCAL. \square

3.1. Interpretation

Let A be an arbitrary L -monoid in an iteration category \mathcal{C} . We give the interpretation of a term as a morphism on A . We first notice that UnCAL is a single sorted system. Hence we interpret the singleton context $\langle\langle \& \rangle\rangle$ as A , and a context $\langle\langle x_1, \dots, x_n \rangle\rangle$ as A^n . We interpret a term $(Y \vdash t : X)$ as a morphism

$$\llbracket t \rrbracket_{\mathcal{C}}^A : A^{|Y|} \rightarrow A^{|X|}$$

in \mathcal{C} . (The super and subscripts of $\llbracket - \rrbracket$ may be omitted hereafter.) The table in Fig. 10 shows the interpretation according to the typing rules. This table is read as, for example in (Com), $Y \vdash s : Z \mapsto g : A^{|Y|} \rightarrow A^{|Z|}$ means that when a term s is interpreted as g , i.e., $\llbracket s \rrbracket = g$, and also $\llbracket t \rrbracket = h$, the interpretation $\llbracket s \diamond t \rrbracket$ is defined to be $g \circ h$. Now, it may be clear why we chose this judgment notation. The source and target contexts of a judgment corresponds to the source and target of a morphism in \mathcal{C} in semantics. This is also a usual principle in categorical type theory. All the abbreviations we introduced in §2.1 are justified by the semantics. For example, why we took the abbreviation $s \times t = \langle s \diamond \pi_1, t \diamond \pi_2 \rangle$ is due to

$$\llbracket \langle s \diamond \pi_{X,Y}^X, t \diamond \pi_{X,Y}^Y \rangle \rrbracket = \langle \llbracket s \diamond \pi_{X,Y}^X \rrbracket, \llbracket t \diamond \pi_{X,Y}^Y \rrbracket \rangle = \langle \llbracket s \rrbracket \circ \pi_{X,Y}^X, \llbracket t \rrbracket \circ \pi_{X,Y}^Y \rangle = \llbracket s \rrbracket \times \llbracket t \rrbracket$$

An (*UnCAL*) *model* A of a theory E is an L -monoid in \mathcal{C} such that for every additional axiom $Y \vdash s = t : X$ of E , $\llbracket s \rrbracket_{\mathcal{C}}^A = \llbracket t \rrbracket_{\mathcal{C}}^A$ holds. Thus if a theory is pure, the notions of

(Mark)	$\frac{Y = \langle\langle y_1, \dots, y_n \rangle\rangle}{Y \vdash y_{i_Y} : \&} \mapsto \pi_i : A^{ Y } \rightarrow A$
(Emp)	$\frac{}{Y \vdash ()_Y : \langle\langle \rangle\rangle} \mapsto !_A^{ Y } : A^{ Y } \rightarrow \mathbf{1}$
(Nil)	$\frac{}{Y \vdash \{\}_Y : \&} \mapsto \eta^A \circ !_A^{ Y } : A^{ Y } \rightarrow \mathbf{1} \rightarrow A$
(Man)	$\frac{}{y_1, y_2 \vdash \wedge_{\langle\langle y_1, y_2 \rangle\rangle} : \&} \mapsto \mu^A : A \times A \rightarrow A$
(Com)	$\frac{Y \vdash s : Z}{X \vdash t : Y} \mapsto g : A^{ Y } \rightarrow A^{ Z }$ $\frac{X \vdash t : Y}{X \vdash s \diamond t : Z} \mapsto h : A^{ X } \rightarrow A^{ Y }$ $\frac{}{X \vdash s \diamond t : Z} \mapsto g \circ h : A^{ X } \rightarrow A^{ Z }$
(Label)	$\frac{\ell \in L}{Y \vdash t : \&} \mapsto \llbracket \ell \rrbracket_L^A : A \rightarrow A$ $\frac{Y \vdash t : \&}{Y \vdash \ell : t : \&} \mapsto g : A^{ Y } \rightarrow A$ $\frac{}{Y \vdash \ell : t : \&} \mapsto \llbracket \ell \rrbracket_L^A \circ g : A^{ Y } \rightarrow A$
(Pair)	$\frac{Y \vdash s : X_1}{Y \vdash t : X_2} \mapsto g : A^{ Y } \rightarrow A^{ X_1 }$ $\frac{}{Y \vdash t : X_2} \mapsto h : A^{ Y } \rightarrow A^{ X_2 }$ $\frac{}{Y \vdash \langle s, t \rangle : X_1 + X_2} \mapsto \langle g, h \rangle : A^{ Y } \rightarrow A^{ X_1 } \times A^{ X_2 }$
(Cyc)	$\frac{Y + X \vdash t : X}{Y \vdash \text{cycle}^X(t) : X} \mapsto g : A^{ Y } \times A^{ X } \rightarrow A^{ X }$ $\frac{}{Y \vdash \text{cycle}^X(t) : X} \mapsto (g)^\dagger : A^{ Y } \rightarrow A^{ X }$
(Def)	$\frac{Y \vdash t : \&}{Y \vdash (x \triangleleft t) : x} \mapsto g : A^{ Y } \rightarrow A$ $\frac{}{Y \vdash (x \triangleleft t) : x} \mapsto g : A^{ Y } \rightarrow A$

Fig. 10. Categorical interpretation $\llbracket - \rrbracket_{\mathcal{C}}^A$

models and L -monoids coincide. We simply say “a model” to mean “a model of E ” if the theory or axioms E we are assuming is clear from the context.

We show another important principle of categorical type theory.

Lemma 3.4. (Substitution as composition) Let A be a model in \mathcal{C} . Under the assumption of Def. 2.3,

$$\llbracket t [\vec{y} \mapsto \vec{s}] \rrbracket = \llbracket t \rrbracket \circ \langle \vec{\llbracket s \rrbracket} \rangle.$$

Proof. By induction on the typing derivation of t . □

As a corollary, renamed terms have the same meaning in \mathcal{C} . Thus, the names of markers are unimportant in the categorical semantics.

Corollary 3.5. Let $Y \vdash t : X$. Consider a renamed term $Y' \vdash t[Y \mapsto Y'] : X$, where $|Y| = |Y'|$. Then $\llbracket t \rrbracket = \llbracket t[Y \mapsto Y'] \rrbracket$ in \mathcal{C} .

Theorem 3.6. (Soundness) Let $(A, \llbracket - \rrbracket_L^A)$ be a model in an iteration category \mathcal{C} . For any theorem $Y \vdash s = t : X$, we have $\llbracket s \rrbracket_{\mathcal{C}}^A = \llbracket t \rrbracket_{\mathcal{C}}^A$.

Proof. By induction on the derivation of proof of $Y \vdash s = t : X$. We check every axiom is sound. The axiom for (sub) is sound by Lemma 3.4. Since \mathcal{C} is an iteration category, the axioms for Parameterised fixed point are sound. Since A is an L -monoid, the axioms for commutative monoid, degenerated bialgebra and deleting trivial cycle are sound. The induction step is routine. \square

3.2. Categorical completeness

We first define an operation that renames markers in the target context, which is need to identify judgments in a categorical semantics.

Definition 3.7. Suppose contexts X, X' with $|X| = |X'|$, and $Y \vdash t : X$. Then a renaming $Y \vdash t \{X \mapsto X'\} : X'$ of target context is inductively defined as follows.

$$\begin{aligned}
t \{&\& \mapsto x'\} &\triangleq (x' \triangleleft t) \quad \text{for } t = y, \{\}_Y, \wedge, (\ell : t) \\
(\)_Y \{&\langle\langle\ \rangle\rangle \mapsto \langle\langle\ \rangle\rangle\} &\triangleq (\)_Y \\
(t_1 \diamond t_2) \{&X \mapsto X'\} &\triangleq (t_1 \{X \mapsto X'\}) \diamond t_2 \\
\langle t_1, t_2 \rangle \{&X_1, X_2 \mapsto X'_1, X'_2\} &\triangleq \langle t_1 \{X_1 \mapsto X'_1\}, t_2 \{X_2 \mapsto X'_2\} \rangle \\
&\text{for } Y \vdash t_1 : X_1, Y \vdash t_2 : X_2 \\
\text{cycle}^X(t) \{&X \mapsto X'\} &\triangleq \text{cycle}^{X'}(t \{X \mapsto X'\}) \\
(x \triangleleft t) \{&x \mapsto x'\} &\triangleq (x' \triangleleft t)
\end{aligned}$$

We establish the completeness of categorical model. Let E be arbitrary UnCAL axioms (regardless of pure or having additional axioms). For a theory generated by E , we construct the classifying category \mathcal{Cl}_E (cf. (Crole, 1993; Hasegawa, 1997a)). As in ordinary categorical type theory, we regard a term as an morphism, namely, we regard $Y \vdash t : X$ as a morphism $t : |Y| \longrightarrow |X|$. More precisely, we identify several terms as the same morphism by using the equality generated by E . We formulate it as follows.

An UnCAL theory E defines an equivalence relation on well-typed terms, which we denote by $=_E$. Suppose contexts $Y = \langle\overrightarrow{y}\rangle, Y' = \langle\overrightarrow{y'}\rangle, X, X'$ with $|Y| = |Y'|, |X| = |X'|$. We identify a term $Y \vdash t : X$ with

$$Y' \vdash t [\overrightarrow{y_Y} \mapsto \overrightarrow{y_{Y'}}] \{X \mapsto X'\} : X'$$

which is a term obtained by bijectively renaming markers of t in the source and target context suitably. Define \approx_E to be the symmetric transitive closure of the union of the equivalence relation $=_E$ and this renaming identification, and write an equivalence class of terms by \approx_E as $[Y \vdash t : X]_E$. To establish categorical completeness, we define the classifying category \mathcal{Cl}_E by taking

- objects: natural numbers $n \in \mathbb{N}$
- morphisms: $[Y \vdash t : X]_E : |Y| \rightarrow |X|$, identity: $\text{id}_{|X|} \triangleq [\text{id}_X]_E : |X| \rightarrow |X|$
- composition: $[Y \vdash s : Z]_E \circ [X \vdash t : Y]_E \triangleq [X \vdash s \diamond t : Z]_E$.

Proposition 3.8. \mathcal{Cl}_E is an iteration category having an L -monoid \mathbf{U} .

Proof. We take

- terminal object: $0 = |\langle\langle\rangle\rangle| \in \mathbb{N}$
- product: the addition $+$ on \mathbb{N}
- projections: $[\pi_{X,Y}^X]_E, [\pi_{X,Y}^Y]_E$
- unit: $\eta \triangleq [\{\}\langle\langle\rangle\rangle]_E : 0 \rightarrow 1$
- counit: $! \triangleq [(\rangle)_{\&}]_E : 1 \rightarrow 0$
- pair: $\langle[s]_E, [t]_E\rangle \triangleq [Y \vdash \langle s, t \rangle : X_1 + X_2]_E$
- Conway: $([Y + X \vdash t : X]_E)^\dagger = [Y \vdash \text{cycle}^X(t) : X]_E$
- monoid object: $U \triangleq 1 = |\langle\langle\&\rangle\rangle| \in \mathbb{N}$
- multiplication: $\mu \triangleq [y_1, y_2 \vdash \wedge_{\langle\langle y_1, y_2 \rangle\rangle} : \&]_E : 1 + 1 \rightarrow 1$
- comonoid: $(U, \Delta, !)$

and $[\ell]_L^U \triangleq [\& \vdash \ell : \& : \&]_E$ for every $\ell \in L$. Then these data satisfy that $\mathcal{C}l_E$ is an iteration category and U is an L -monoid because of the axioms **AxG**. \square

Remark 3.9. As the definitions of $\mathcal{C}l_E$, pair of arrows, and the multiplication shown in the above proof, our choice of the modified notation of **UnCAL** actually came from this categorical structure. Note that $\mathcal{C}l_E$ is now the opposite category of an *iteration theory* by Bloom and Ésik (1993) (N.B. a “theory” means here a Lawvere theory). \square

Lemma 3.10. $[\![t]\!]^U = [t]_E$.

Proof. By induction on the typing derivation of t . For the case of (Nil), Lemma 3.4 is used. For the case of (Label), (sub) axiom is used. Other cases are routine. \square

Now U is a particular model called a generic model in $\mathcal{C}l_E$.

Proposition 3.11. (Generic model) The L -monoid U in $\mathcal{C}l_E$ is a model of E .

Proof. For every $(X \vdash s = t : Y)$ in E , $[\![s]\!]^U = [s]_E = [t]_E = [\![t]\!]^U$ by Lemma 3.10. \square

Theorem 3.12. (Categorical soundness and completeness) $Y \vdash s = t : X$ is derivable from E (including **AxG**) in **EL-UnCAL** iff $[\![s]\!]_{\mathcal{C}}^A = [\![t]\!]_{\mathcal{C}}^A$ holds for all iteration categories \mathcal{C} and all models A of E in \mathcal{C} .

Proof. Soundness has been established in Thm. 3.6. Suppose the assumption of the theorem. Then in particular for the classifying category $\mathcal{C}l_E$ and the generic model U , $[\![s]\!]_{\mathcal{C}l_E}^U = [\![t]\!]_{\mathcal{C}l_E}^U$ holds. Hence $[s]_E = [t]_E$ meaning that $s = t$ is derivable from E . \square

Theorem 3.13. For any model A of E in an iteration category \mathcal{C} , there exists a unique iteration functor $\Psi^A : \mathcal{C}l_E \longrightarrow \mathcal{C}$ such that $\Psi^A(U) = A$, and it preserves L -monoid structures. Pictorially, it is expressed as the right picture, where Tm denotes the set of all well-typed terms.

$$\begin{array}{ccc} \text{Tm} & \xrightarrow{[\![-]\!]^U} & \mathcal{C}l_E \\ \downarrow [\![-]\!]^A & \swarrow \Psi^A & \\ \mathcal{C} & & \end{array}$$

Proof. We write simply Ψ for Ψ^A . Since Ψ preserves L -monoids, we have $[\![t]\!]^A = \Psi([\![t]\!]^U) = \Psi([t]_E)$ for any term t , hence the mapping Ψ is required to satisfy

$$\begin{array}{ll} \Psi([y_i]_E) = \pi_i & \Psi([s \diamond t]_E) = \Psi([s]_E) \circ \Psi([t]_E) \\ \Psi([\langle \rangle]_E) = !_{A|Y|} & \Psi([\ell : t]_E) = [\ell]_L^A \circ \Psi([t]_E) \\ \Psi([\{\}]_E) = \eta^A \circ !_{A|Y|} & \Psi([\langle s, t \rangle]_E) = \langle \Psi([s]_E), \Psi([t]_E) \rangle \\ \Psi([\wedge]_E) = \mu^A & \Psi([\text{cycle}(t)]_E) = (\Psi([t]_E))^\dagger \\ \Psi([x \triangleleft t]_E) = \Psi([t]_E) & \end{array}$$

This means that Ψ is an iteration functor that sends the L -monoid U to the L -monoid A . Such Ψ is uniquely determined by these equations because $(A, \llbracket - \rrbracket^A)$ is a model. \square

So, the interpretation $\llbracket - \rrbracket^A$ determines an iteration functor Ψ^A . What is Ψ^A from the viewpoint of UnCAL’s graph transformation? The composite $\Psi^A(\llbracket - \rrbracket^U)$ sends a directed edge in an input term to a morphism in \mathcal{C} whose source and target nodes are correspondingly mapped. This idea is more precisely pursued in the next section.

4. Characterisation of Structural and Primitive Recursion

In this section, we tackle to model the computation mechanism of UnCAL “structural recursion on graphs” using our categorical semantics.

4.1. Overview

A structural recursive function is defined by the syntax

$$\begin{aligned} \mathbf{sfun} \ f(\ell_1:t) &= e_{\ell_1} \\ &\dots \\ f(\ell_n:t) &= e_{\ell_n} \end{aligned} \tag{1}$$

where ℓ_1, \dots, ℓ_n are disjoint and cover all labels in L . We assume that the number n of clauses is finite. Some ℓ_i may be meta-variables and the case analysis will be given meta-theoretically as in Example 4.1 below. Each e_{ℓ_i} is an UnCAL term extended with the form $g(s)$ where g is a label. To concentrate on the essential part of modelling structural recursion, in this paper, we assume that every e_{ℓ_i} does not involve other function calls $h(t)$ where h is defined by other structural recursive function defined by \mathbf{sfun} -definition. But we can use other functions defined by additional axioms, as in Example 4.1.

In the original formulation, given an input graph, the graph algorithm computes a result graph using the definition (1). This relation between inputs and results becomes a function f , which satisfies the equations (Buneman et al., 2000, Prop. 3):

$$\begin{aligned} f(y_i) &= y_i & f(x \triangleleft t) &= (x \triangleleft f(t)) & f(\ell:t) &= e_\ell & \dots (\star) \\ f(()) &= () & f(s \cup t) &= f(s) \cup f(t) & f(s \diamond t) &= f(s) \diamond f(t) & \dots (\bowtie) \\ f(\{\}) &= \{\} & f(\langle s, t \rangle) &= \langle f(s), f(t) \rangle & f(\text{cycle}(t)) &= \text{cycle}(f(t)) & \dots (\bowtie) \end{aligned} \tag{2}$$

when e_ℓ does not depend on t . We intend that the part (\star) represents a family of equations for all labels ℓ as (1). Here “ e_ℓ depends on t ” means that e_ℓ contains t other than the form $f(t)$. This is understandable naturally, as the example **f2** in Introduction recurses the term \mathbf{t}_G structurally. Combining the categorical viewpoint we have investigated, f can be understood as a functor because it preserves \diamond in (\bowtie) . Moreover, f preserves cycle and cartesian products, hence it can be regarded as a traced cartesian functor. Importantly, Buneman et al. observed that the above nine equations hold only when e_ℓ does not depend on t , such as **f2**.

The case where e_ℓ depends on t requires more careful thought. Two equations marked

(\bowtie) do not hold in this case (and other seven equations do hold). Crucially, **f1** in Introduction is already in this case, where **T** appears as not of the form **f1**(t). The following example shows why (\bowtie) do not hold.

Example 4.1. We suppose the labels **true** and **false**. We abbreviate the terms **true**: $\{\}$ and **false**: $\{\}$ as simply **true** and **false**. We first define the additional axioms

$$\left\{ \begin{array}{l} \text{head-a?}(\{\}) = \{\} \\ \text{head-a?}(\text{a:}t) = \text{true} \\ \text{head-a?}(\ell:t) = \{\} \quad \text{for } \ell \neq \text{a} \end{array} \right\}$$

for a predicate **head-a?** that checks whether the head is **a**. Since this is not a recursive definition, we set it as axioms. The UnCAL axioms E we now assume is the union of **AxG** and the additional axioms. We define the recursive function **aa?** that tests whether the argument contains “**a:a:**”.

$$\begin{aligned} \text{sfun aa?}(\text{a:}t) &= \text{head-a?}(t) \\ \text{aa?}(\ell:t) &= \text{aa?}(t) \quad \text{for } \ell \neq \text{a} \end{aligned}$$

The right-hand side **head-a?**(t) of the first clause depends on t . Then we have the inequalities:

$$\begin{aligned} \text{aa?}(\text{a:}\&\diamond\text{a:}\{\}) &= \text{aa?}(\text{a:a:}\{\}) = \text{true} \\ &\neq \{\} = \{\}\diamond\{\} = \text{aa?}(\text{a:}\&) \diamond \text{aa?}(\text{a:}\{\}) \\ \text{aa?}(\text{cycle}(\text{a:}\&)) &= \text{aa?}(\text{a:a:}\text{cycle}(\text{a:}\&)) = \text{true} \\ &\neq \{\} = \text{cycle}(\{\}) = \text{cycle}(\text{aa?}(\text{a:}\&)) \end{aligned}$$

This means that f does not preserve **cycle** in general, and even is not functorial. Thus the categorical view seems not helpful to understand this pattern of recursion.

This pattern of recursion is similar to the case of primitive recursion on an algebraic data type. Since structural recursion is a generalisation of primitive recursion, primitive recursion can be encoded as a structural recursion. The technique of paramorphism (Meertens, 1992) in functional programming is a way to represent primitive recursion in terms of “fold”. Using a similar idea, we will show that the case e_ℓ depends on t can also be derivable from the universality.

4.1.1. Aims. In this section, we will show the following.

- (i) We give a way to generate mathematically a function Ψ (resp. Υ) on UnCAL terms from the syntactic definition (1) of a function f for structural recursion in §4.2 (resp. primitive recursion in §4.3).
- (ii) We clarify what equations are satisfied for Ψ (resp. Υ). We call these equations *the characterisation* of structural recursive f (3) (resp. primitive recursive (6)).
- (iii) We show that the characterisation *uniquely determines* a function.

Throughout this section, we suppose UnCAL axioms E , (i.e. E is the union of **AxG** and additional axioms) as stated in §3.2. See also §2.3. We may use additional axioms to define built-in and utility functions as Example 4.1.

4.2. Structural recursion on graphs

We first characterise structural recursion on graphs for the case that e_ℓ may involve the form $f(t)$ in (1), but does not involve t solely, such as the function **f2** in Introduction. To formally formulate the form $f(t)$, we extend our syntax to have the construct of function terms given by the following typing rule.

$$(\text{Fun}) \frac{f \in L \quad Y \vdash t : \&}{Y \vdash f(t) : \&}$$

We assume that L contains also function symbols. The categorical semantics of this is given by $\llbracket f(t) \rrbracket = \llbracket f \rrbracket_L \circ \llbracket t \rrbracket$. The idea of this semantics is to represent the form $f(t)$ at the level of categorical arrows. Note that this does not give the semantics of recursive calls. It is no problem because the term of recursive call $f(t)$ is vanished in modelling the structural recursion f (cf. the construction of e'_ℓ below).

4.2.1. Assumptions for structural recursion. Suppose the definition of structural recursive function f is of the form (1). Let e_ℓ be the right-hand side of a clause in (1). We assume that it is well-typed as

$$Y \vdash e_\ell : W$$

where $W = \langle x_1, \dots, x_k \rangle$ (i.e. there exist suitable contexts W, Y for e_ℓ) and

$$k \triangleq |W|.$$

(i) We assume that a term $f(t)$ possibly appears in e_ℓ and is of the type

$$Y \vdash f(t) : W$$

(ii) We assume that all the right-hand sides e_{ℓ_i} in (1) have the same k and source and target contexts Y, W . **We use this number k throughout this section.**

We define e'_ℓ to be a term obtained from e_ℓ , by replacing every $f(t)$ with $\langle x_1 \triangleleft x_1, \dots, x_k \triangleleft x_k \rangle$ of type W and assume

$$W \vdash e'_\ell : W.$$

By construction, $Y \vdash e_\ell = e'_\ell \diamond f(t) : W$ holds.

4.2.2. The case $k = 1$. We first examine the simplest case $k = |W| = 1$ to get intuition of our categorical characterisation of structural recursion. The idea of constructing e'_ℓ above is that since e_ℓ involves several $f(t)$'s, we first separate these from e_ℓ . Now e'_ℓ is a “skeleton” of e_ℓ where every occurrence of the recursive call $f(t)$ is replaced with the marker $\&$, considered as a hole. Then we model the execution of the structural recursion (1) as the operation that replaces all labelled edges “ ℓ ” in a term with e'_ℓ with a suitable composition. This is achieved by taking the L -monoid $M \triangleq (U, \llbracket - \rrbracket_L^M)$ in \mathcal{Cl}_E by

$$\llbracket \ell \rrbracket_L^M \triangleq [\& \vdash e'_\ell : \&]_E : U \rightarrow U,$$

which is moreover a model of E .

Remark 4.2. For example, in case of the example **f2** in Introduction defined by

$$\mathbf{sfun} \text{ f2}(\ell:t) = \mathbf{a:f2}(t)$$

we take $e_\ell = \mathbf{a:f2}(t)$, $e'_\ell = \mathbf{a:\&}$, $\llbracket \ell \rrbracket_L^M = [\& \vdash \mathbf{a:\& : \&}]_E$. \square

Notice that M is different from the L -monoid $U = (U, \llbracket - \rrbracket_L^U)$ of the generic model, where $\llbracket \ell \rrbracket_L^U = [\ell:\&]_E$. Now we consider the mapping from the generic model to the model M . In the situation of Theorem 3.13, the unique iteration functor $\Psi^U : \mathcal{C}l_E \rightarrow \mathcal{C}l_E$ exists and it maps the L -monoid U to the L -monoid M . Thus, what the functor Ψ^U does is that every morphism for ℓ (coming from a labelled edge ℓ in a term) is replaced with e'_ℓ as

$$\begin{array}{ccc} U & & U \\ \ell \uparrow & \mapsto & \uparrow e'_\ell \\ U & & U \end{array}$$

which is actually we wanted to model. Namely, we model the meaning of the syntactically given function f as a functor Ψ^U .

More precisely, it is modelled as follows. We may omit the superscript of Ψ , hereafter. Since U is a model, the choice of a representative in an equivalence class applied to Ψ is irrelevant, i.e., if $s \approx_E t$, then $\Psi([s]_E) = \Psi([t]_E)$. This means that it is safe to write simply $\Psi(t)$ for a suitably renamed term t avoiding name clash, and then it gives a syntactic translation on UnCAL terms by the unique (arrow part) function

$$\Psi : \mathcal{C}l_E(U^{|Y|}, U^{|X|}) \rightarrow \mathcal{C}l_E(U^{|Y|}, U^{|X|})$$

on terms of source context Y and target context X , characterised as

$$\begin{array}{lll} \Psi(y_i) = y_i & \Psi(\lambda) = \lambda & \Psi(\ell:t) = e'_\ell \diamond \Psi(t) \\ \Psi(()) = () & \Psi(x \triangleleft t) = \Psi(t) & \Psi(s \diamond t) = \Psi(s) \diamond \Psi(t) \\ \Psi(\{\}) = \{\} & \Psi(\langle s, t \rangle) = \langle \Psi(s), \Psi(t) \rangle & \Psi(\text{cycle}(t)) = \text{cycle}(\Psi(t)) \end{array} \quad (3)$$

Then one can clearly see that Ψ traverses a term in a structural recursive manner. Note that the case for $(x \triangleleft t)$, it actually translates equivalence classes as

$$\Psi([Y \vdash (x \triangleleft t) : \&]_E) = \Psi([Y \vdash t : \&]_E) = \Psi([t]_E).$$

Here comes an important point. This pattern Ψ derived from the universality in Thm. 3.13 is exactly the same as what Buneman et al. called the “*structural recursion on graphs*” for the case that e_ℓ does not depend on t (Buneman et al., 2000, Proposition 3). Actually, we could make the characterisation more precise than the original and the subsequent developments (Buneman et al., 2000; Hidaka et al., 2010, 2011, 2013a; Asada et al., 2013).

- (i) Buneman et al. stated only that (3) is a *property* (Buneman et al., 2000, Prop. 3) of a “structural recursive function on graphs” defined by the algorithms. This means that there could be many functions that satisfy the property. Our result of the universality stated in Theorem 3.13 provides more precise information. A function satisfying the characterisation (3) *uniquely* determines a function Ψ , and there are no functions that satisfy the characterisation other than Ψ .

- (ii) The characterisation mathematically clarifies what are the *structures* of “structural recursion on graphs”. The structures preserved by it are *iteration category* and *L-monoid* structures.

Example 4.3. (Replace all labels with a)

$$\text{sfun } f3(\ell:t) = a:f3(t)$$

Example of execution is:

```
f3( people:{ population:425017:{}, ethnicGroup:"Celtic":{},
           ethnicGroup:"Portuguese":{}, ethnicGroup:"Italian":{} })
↪ a:{ a:a:{}, a:a:{}, a:a:{} }
```

Define the particular L -monoid $(U, \llbracket - \rrbracket_L)$ in $\mathcal{C}\ell_E$ by $\llbracket \ell \rrbracket_L = [\& \vdash a:\& : \&]_E$. Then the unique iteration functor $\Psi : \mathcal{C}\ell_E \rightarrow \mathcal{C}\ell_E$ gives the structural recursive function defined by $f3$. Actually, it computes

```
Ψ( people:{ population:425017:{}, ethnicGroup:"Celtic":{},
           ethnicGroup:"Portuguese":{}, ethnicGroup:"Italian":{} })
= a:{ a:a:{}, a:a:{}, a:a:{} }
```

4.2.3. **The case $k \geq 1$.** Again consider the setting of §4.2.1. The method we have taken can be generalised to the case for any $k \geq 1$, i.e. e_ℓ is of the type

$$Y \vdash e_\ell : x_1, \dots, x_k$$

This is merely by taking U^k (instead of U) as a model. Every object U^k forms a monoid with unit and multiplication given by

$$\begin{aligned} \nu^{U^k} &: \mathbf{1} \rightarrow U^k \\ \nu^{U^k} &\triangleq [\langle \{\} \rangle_0, \dots, \{\} \rangle_E \\ \mu^{U^k} &: U^k \times U^k \rightarrow U^k \\ \mu^{U^k} &\triangleq [x_1, \dots, x_k, y_1, \dots, y_k \vdash \langle \{x_1\} \cup \{y_1\}, \dots, \{x_k\} \cup \{y_k\} \rangle : \&_1, \dots, \&_k]_E \end{aligned}$$

It forms the L -monoid $M \triangleq (U^k, \llbracket - \rrbracket_L^{U^k})$ by

$$\llbracket \ell \rrbracket_L^M \triangleq [W \vdash e'_\ell : W]_E : U^k \rightarrow U^k$$

which is moreover a model. Thus, the unique iteration functor $\Psi^{U^k} : \mathcal{C}\ell_E \rightarrow \mathcal{C}\ell_E$ exists such that $\Psi^{U^k}(U) = U^k$ and preserves L -monoid structures. Similarly to the case $k = 1$, it gives the unique function on UnCAL terms

$$\Psi : \mathcal{C}\ell_E(U^{|Y|}, U^{|X|}) \rightarrow \mathcal{C}\ell_E(U^{k \cdot |Y|}, U^{k \cdot |X|}) \quad (4)$$

which is characterised as

$$\begin{aligned} \Psi(y_i) &= \langle y_i^1, \dots, y_i^k \rangle & \Psi(\text{and}) &= \mu^{U^k} & \Psi(\ell:t) &= e'_\ell \diamond \Psi(t) \\ \Psi(()) &= () & \Psi(x \triangleleft t) &= \Psi(t) & \Psi(s \diamond t) &= \Psi(s) \diamond \Psi(t) \\ \Psi(\{\}) &= \langle \{\}, \dots, \{\} \rangle & \Psi(\langle s, t \rangle) &= \langle \Psi(s), \Psi(t) \rangle & \Psi(\text{cycle}(t)) &= \text{cycle}(\Psi(t)) \end{aligned} \quad (5)$$

Note that Ψ returns a term in a context Y^1, \dots, Y^k , where $Y^1 = \langle\langle y_1^1, \dots, y_m^1 \rangle\rangle$, $Y^2 = \langle\langle y_1^2, \dots, y_m^2 \rangle\rangle, \dots$, and the superscripts indicate i -th copy of Y ($1 \leq i \leq k$). As in the case

of $k = 1$, the equations (5) *uniquely determine* the structural recursive function Ψ . For this unique function Ψ , we will write

$$\mathbf{srec}((e'_\ell)_{\ell \in L}) \quad \text{or simply} \quad \mathbf{srec}(e')$$

for $e' = (e'_\ell)_{\ell \in L}$.

This is our semantical explanation of “structural recursion on graphs”. As a bonus, the characterisation reveals minor errors in the previous definitions of structural recursion in the literature (Buneman et al., 2000; Hidaka et al., 2010, 2011). The case for $\{\}$ must be $\Psi(\{\}) = \langle \{\}, \dots, \{\} \rangle$. But in the literature mentioned above, this was $\Psi(\{\}) = \{\}$ (N.B. $\langle \{\}, \dots, \{\} \rangle \neq \{\}$). This shows the importance of identification of the mathematical structure (i.e. a monoid U^k) to provide a right definition. The papers (Hidaka et al., 2013a; Asada et al., 2013) correctly handle this case by considering a suitable monoid as our formulation.

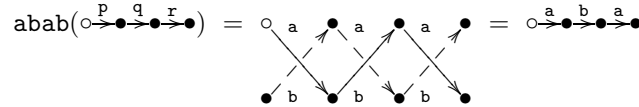
Example 4.4. ((Hidaka et al., 2010) abab) We demonstrate structural recursion on graphs of the case $k = 2$. The function **abab** changes all edges of even distance from the root to **a**, odd distance edges to **b**. It is defined by

$$\mathbf{sfun} \text{ abab}(\ell:t) = \langle (z1 \triangleleft \mathbf{a}:z2), (z2 \triangleleft \mathbf{b}:z1) \rangle \diamond \text{abab}(t)$$

Example of execution is:

$$z1 @ \text{abab}(\mathbf{p:q:r:\{\}}) \rightsquigarrow \mathbf{a:b:a:\{\}}.$$

The original execution process was hard to understand, because the structural recursion is officially defined by the graph algorithms (Buneman et al., 2000). (Hidaka et al., 2010) gave an informal explanation how it is executed using the following figure



but it is still mysterious for outsiders. What happens exactly in the computation process? Now we take e'_ℓ to be the term

$$z_1, z_2 \vdash \langle (z_1 \triangleleft \mathbf{a}:z_2), (z_2 \triangleleft \mathbf{b}:z_1) \rangle : z_1, z_2$$

This is the case $k = 2$. Here we give a formal *equational proof* using our characterisation of structural recursion and the theorems (sub) and (asso \diamond):

$$\begin{aligned} & z_1 \diamond \mathbf{srec}(e'_\ell)(\mathbf{p:q:r:\{\}}) \\ = & z_1 \diamond (e'_\ell \diamond e'_\ell \diamond e'_\ell \diamond \langle \{\}, \{\} \rangle) = z_1 \diamond (e'_\ell \diamond \langle (z_1 \triangleleft \mathbf{a}:z_2), (z_2 \triangleleft \mathbf{b}:z_1) \rangle \\ & \quad \diamond \langle (z_1 \triangleleft \mathbf{a:\{\}}, (z_2 \triangleleft \mathbf{b:\{\}}) \rangle) \\ = & (z_1 \diamond e'_\ell) \diamond \langle (z_1 \triangleleft \mathbf{a:b:\{\}}, (z_2 \triangleleft \mathbf{b:a:\{\}}) \rangle = \mathbf{a:b:a:\{\}}. \end{aligned}$$

This *equational proof* has not been possible so far (Buneman et al., 2000; Hidaka et al., 2010, 2011, 2013a; Asada et al., 2013).

4.3. Primitive recursion on graphs

4.3.1. Assumptions for primitive recursion. Suppose the definition of structural recursive function f is of the form (1). Let e_ℓ be the right-hand side of a clause in (1), which is well-typed as

$$\vdash e_\ell : W$$

where $W = \langle\langle x_1, \dots, x_k \rangle\rangle$ and

$$k \triangleq |W|.$$

(i) We assume that terms $f(t), t$ possibly appear in e_ℓ and are of the types

$$\vdash f(t) : W \quad \vdash t : \&$$

(ii) We assume that all the right-hand sides e_{ℓ_i} in (1) have the same k , the empty source context and the target context W .

We define e'_ℓ to be a term obtained from e_ℓ , by replacing every $f(t)$ with $\langle x_1 \triangleleft x_1, \dots, x_k \triangleleft x_k \rangle$ of type W , and every t (not in the form $f(t)$) with $\&$, and assume

$$W, \& \vdash e'_\ell : W.$$

By construction, $\vdash e_\ell = e'_\ell \diamond \langle f(t), t \rangle : W$ holds.

The marker $\&$ in e'_ℓ is regarded as a hole which will be filled with t , and $\langle x_1 \triangleleft x_1, \dots, x_k \triangleleft x_k \rangle$ is regarded as other holes filled with $f(t)$. We call this type of recursion *primitive recursion on graphs*, because it is similar to the case $f(S(n)) = e(f(n), n)$ of primitive recursion on natural numbers, which can use both n and $f(n)$ at the right-hand side. Now we take another monoid $U^k \times U$ as an UnCAL model. $U^k \times U$ forms a monoid as in the case of U^k , moreover a L -monoid by

$$[\![\ell]\!]_{L}^{U^k \times U} : U^k \times U \rightarrow U^k \times U, \quad [\![\ell]\!]_{L}^{U^k \times U} \triangleq [W, \& \vdash \langle e'_\ell, (\ell; \&) \rangle : W, \&]_E.$$

Since $U^k \times U$ is a model, there exists the unique (arrow part) function

$$\Psi : \mathcal{C}\ell_E(U^{|Y|}, U^{|X|}) \rightarrow \mathcal{C}\ell_E(U^{(k+1) \cdot |Y|}, U^{(k+1) \cdot |X|})$$

by Thm. 3.13. Finally, we define the function

$$\Upsilon : \mathcal{C}\ell_E(\mathbf{1}, U^{|X|}) \rightarrow \mathcal{C}\ell_E(\mathbf{1}, U^{k \cdot |X|}); \quad \Upsilon(t) \triangleq \pi \diamond \Psi(t)$$

where we take $Y = \langle\langle \rangle\rangle$ for Ψ and π is a projection that projects $U^{k \cdot |X|}$ -components from $U^{(k+1) \cdot |X|}$, given by

$$X^1, \dots, X^k, X \vdash \pi \triangleq \langle X^1, \dots, X^k \rangle : X^1, \dots, X^k,$$

and $X^i = \langle\langle x_1^i, \dots, x_m^i \rangle\rangle$, where the superscripts indicate that it is the i -th copy taken from X ($1 \leq i \leq k$). Then $\Psi(t) = \langle \Upsilon(t), t \rangle$ holds for any term $\vdash t : \&$. Hence using (3), Υ is the *unique function* satisfying

$$\begin{aligned} \Upsilon(\{s\} \cup \{t\}) &= \{\Upsilon(s)\} \cup \{\Upsilon(t)\} & \Upsilon(\ell; t) &= e'_\ell \diamond \langle \Upsilon(t), t \rangle \\ \Upsilon((\)_0) &= (\)_0 & \Upsilon(x \triangleleft t) &= \Upsilon(t) & \Upsilon(s \diamond t) &= \pi \diamond (\Psi(s) \diamond \Psi(t)) \\ \Upsilon(\{s\}_0) &= \{\{s\}_0, \dots, \{s\}_0\} & \Upsilon(\langle s, t \rangle) &= \langle \Upsilon(s), \Upsilon(t) \rangle & \Upsilon(\text{cycle}(t)) &= \pi \diamond \text{cycle}(\Psi(t)) \end{aligned} \tag{6}$$

This Υ completely characterises *what Buneman et al. called the structural recursion on graphs for the case that e_ℓ depends on t* . This result provides more precise information than the existing results.

- (i) The equations (6) are a unique characterisation of primitive recursion on UnCAL graphs.
- (ii) Buneman et al. showed by a counterexample that $f(s \diamond t) = f(s) \diamond f(t)$ and $f(\text{cycle}(t)) = \text{cycle}(f(t))$ do not hold in general for a primitive recursive f . They did not analyse the precise reason of it and what equations *do* hold for them. Now we could give the laws for \diamond and cycle in (6).
- (iii) These laws can be read as how to compute primitive recursion Υ for the cases for $s \diamond t$ and $\text{cycle}(t)$ using the structural recursion Ψ (N.B. not Υ). But this way is not mandatory. Since Υ respects the equality \approx_E , taking any term u which is equivalent (w.r.t. \approx_E) to $s \diamond t$ (resp. $\text{cycle}(t)$), and next computing $\Upsilon(u)$ is also possible. The following example illustrates this situation. In any case, the laws of Υ for \diamond and cycle in (6) hold.

Note that Υ is a set-theoretical total function, hence it is completely determined (in other words, terminating). For this unique function Υ , we will write

$$\mathbf{prec}((e'_\ell)_{\ell \in L}) \quad \text{or simply} \quad \mathbf{prec}(e').$$

Example 4.5. We model the primitive recursive function $\mathbf{aa?}$ given in Example 4.1. Take an L -monoid

$$\begin{aligned} \llbracket \mathbf{a} \rrbracket_L &= [x, \& \vdash \langle \mathbf{head-a?}(\&) , \mathbf{a}:\& \rangle : x, \&]_E \\ \llbracket \ell \rrbracket_L &= [x, \& \vdash \langle x , \ell:\& \rangle : x, \&]_E \text{ for } \ell \neq \mathbf{a}. \end{aligned}$$

We can compute $\mathbf{aa?}(\text{cycle}(\mathbf{a}:\&))$ as follows by using the law for cycle in (3).

$$\begin{aligned} \Upsilon(\text{cycle}(\mathbf{a}:\&)) &= \pi \diamond \text{cycle}(\Psi(\mathbf{a}:\&)) \\ &= \pi \diamond \text{cycle}(\langle \mathbf{head-a?}(\&) , \mathbf{a}:\& \rangle) \\ &= \pi \diamond \langle \mathbf{head-a?}(\&) , \mathbf{a}:\& \rangle \diamond \langle \mathbf{head-a?}(\&) , \mathbf{a}:\& \rangle \diamond \text{cycle}(\mathbf{a}:\&) \\ &= \pi \diamond \langle \mathbf{head-a?}(\mathbf{a}:\&) , \mathbf{a}:\mathbf{a}:\& \rangle \diamond \text{cycle}(\mathbf{a}:\&) = \mathbf{true}. \end{aligned}$$

There is another way to compute it by firstly unfolding the argument using the fixed point law (fix) without relying the structural recursive function Ψ . Now $e' = \mathbf{head-a?}(\&)$.

$$\begin{aligned} \Upsilon(\text{cycle}(\mathbf{a}:\&)) &= \Upsilon(\mathbf{a}:\mathbf{a}:\text{cycle}(\mathbf{a}:\&)) \\ &= e' \diamond \langle e' \diamond \langle \Upsilon(\text{cycle}(\mathbf{a}:\&)), \text{cycle}(\mathbf{a}:\&) \rangle , \mathbf{a}:\text{cycle}(\mathbf{a}:\&) \rangle \\ &= \mathbf{head-a?}(\mathbf{a}:\text{cycle}(\mathbf{a}:\&)) = \mathbf{true} \end{aligned}$$

In summary, taking a suitable L -monoid, we could model various types of recursion principle uniformly by the universality of the generic model in the categorical semantics of UnCAL.

4.4. Application to the fusion law

We show an application of our characterisation to an optimisation for structural recursion, known as *fusion law*. The following theorem is called a fusion law, which states that the composition of two structural recursive functions can be expressed as a single one. It means that recursing an input term *only once* is enough (at the right-hand side), rather than twice (i.e. at the left-hand side, firstly **srec** recurses an input and secondly h recurses the result), hence it provides an optimisation.

Theorem 4.6. Suppose that $\& \vdash e_\ell : \&$ and $X \vdash d_\ell : X$ are terms for every $\ell \in L$. If a structural recursive function h satisfies $h(e_\ell) = d_\ell$ for all $\ell \in L$, then

$$h \circ \mathbf{srec}(e) = \mathbf{srec}(d). \quad (7)$$

where $e = (e_\ell)_{\ell \in L}$, $d = (d_\ell)_{\ell \in L}$.

This has been proved in (Buneman et al., 2000, Theorem 4). The original proof was quite involved, which spent 3.5 pages long. It was proved by induction on terms and using a technical lemma lifting an equation on finite trees to that on infinite graphs. This involved proof method was unavoidable because in the original formulation,

- graphs are the basic data of the formulation,
- terms denote only a subset of the set of all graphs, and
- infinite graphs are allowed as possible graphs (but no infinite terms are allowed),
- thus terms are not enough to capture all graphs (cf. discussion §1.3.2).
- The statement (7) was actually the statement on finite and infinite graphs and it was not only on finite terms,
- so the technical lifting lemma from finite trees to infinite trees was necessary.

Similar fusion laws were stated in (Hidaka et al., 2013a) for an ordered variant of UnCAL but no proof was given. Because the formulation in Hidaka et al. (2013a) follows the original formulation (Buneman et al., 2000), the formal proofs for the fusion laws, if exist, would have the same problems.

Our algebraic formulation in this paper cleanly avoids these problems. We can now give a much simpler and conceptual proof, which does not even rely on induction. Because we do not use graphs at the level of syntactic formulation, now the statement (7) is actually a statement on UnCAL terms. The proof proceeds as follows.

Proof. Let $k \triangleq |W|$. The proof is due to the following diagram, which commutes by Thm. 3.13. Here $\mathbf{Tm}(Y, X)$ is the set of all well-typed terms having the source context X and the target context Y , Ψ^A, Ψ^B are the arrow part functions of functors, and $\mathcal{C}l_E(\mathbf{U}^{|Y|}, \mathbf{U}^{|X|})$ is a hom-set, cf. (4).

$$\begin{array}{ccc}
\text{Tm}(Y, X) & \xrightarrow{\llbracket - \rrbracket^U} & \mathcal{Cl}_E(U^{|Y|}, U^{|X|}) \\
\llbracket - \rrbracket^A \downarrow & \Psi^A \swarrow & \\
\mathcal{Cl}_E(U^{|Y|}, U^{|X|}) & & \\
h \downarrow & \Psi^B \swarrow & \\
\mathcal{Cl}_E(U^{k \cdot |Y|}, U^{k \cdot |X|}) & &
\end{array}$$

It expresses that the composition of two iteration functors Ψ^A and h , which both preserves L -monoids, is again an iteration functor Ψ^B that preserves L -monoids, and such Ψ^B exists uniquely by the universality.

More precisely, Let $A = (U, \llbracket - \rrbracket_L^A)$, $B = (U^k, \llbracket - \rrbracket_L^B)$ be models of E in \mathcal{Cl}_E . Suppose that h is an iteration functor such that $h(U) = U^k$ and preserves L -monoid structures. Then by Thm. 3.13, two iteration functors

$$\Psi^A = \mathbf{srec}(e), \quad \Psi^B = \mathbf{srec}(d)$$

such that $\Psi^A(U) = U$, $\Psi^B(U) = U^k$, Ψ^A maps the L -monoid U to the L -monoid A , and Ψ^B maps the L -monoid U to the L -monoid B , exist uniquely and the above diagram commutes. Observe that to give an iteration functor h sending the L -monoid A to B is equivalent to giving a structural recursive function h , satisfying

$$h(\llbracket \ell \rrbracket_L^A) = \llbracket \ell \rrbracket_L^B \text{ for all } \ell \in L.$$

This is exactly the condition stated in Thm. 4.6, hence we have proved the fusion law. \square

The fusion law for the case of primitive recursion (Buneman et al., 2000) is

$$\mathbf{prec}(d) \circ \mathbf{prec}(e) = \mathbf{prec}(\mathbf{prec}(d)(\mathbf{prec}(e)(\ell : \&))_{\ell \in L})$$

So we prove:

$$(\mathbf{prec}(d) \circ \mathbf{prec}(e))(s) = \mathbf{prec}(\mathbf{prec}(d)(\mathbf{prec}(e)(\ell : \&))_{\ell \in L})(s)$$

This is proved straightforwardly by induction on the typing derivation of s and the characterisation (6). The difference from the original proof is that now we do not rely on a technical and involved lemma lifting an equation on finite trees to that on infinite graphs. The use of induction on s is validated by the result that **AxG** completely characterises bisimulation of **UnCAL** graphs, which will be proved in the next section.

5. Completeness for Bisimulation

In this section, we prove that our axioms **AxG** completely characterise the extended bisimulation used in the original formulation of **UnCAL**. Namely, **AxG** is a complete syntactic axiomatisation of the original bisimulation equality on **UnCAL** graphs. We prove it by connecting it with the algebraic axiomatisations of bisimulation by Bloom and Ésik (Bloom and Ésik, 1993; Ésik, 2000, 2002). We first review their characterisation result using the notion of μ -terms by following the formulation in (Ésik, 2002, §2.5).

5.1. Equational logic for μ -terms

Let Σ be a signature, i.e. a set of function symbols equipped with arities. We define μ -terms by

$$t ::= x \mid f(t_1, \dots, t_n) \mid \mu x. t,$$

where x is a variable, and f is an n -ary function symbol in Σ . For a set V of variables, we denote by $\mathcal{T}_\mu(V)$ the set of all μ -terms generated by V . We also define the μ -notation on vectors $\mu(x_1, \dots, x_n).(t_1, \dots, t_n)$ by induction on n : if $n = 1$, it is $\mu x_1.t_1$. If $n > 1$, we define

$$\mu(x_1, \dots, x_n).(t_1, \dots, t_n) \triangleq (\mu(x_1, \dots, x_{n-1}).s, \mu x_n.t_n[\mu(x_1, \dots, x_{n-1}).s/(x_1, \dots, x_{n-1})])$$

where $s = (t_1, \dots, t_n)[\mu x_n.t_n/x_n]$. The notation $[t/(x_1, \dots, x_n)]$ means

$$[(\pi_1 t, \dots, \pi_n t)/(x_1, \dots, x_n)]$$

where $\pi_i(x_1, \dots, x_n) = x_i$. We now regard each label $\ell \in L$ as an unary function symbol and consider the signature $\Sigma = L \cup \{0^{(0)}, +^{(2)}\}$ (the superscripts denote the arities).

5.1.1. Axioms for bisimulation. We assume the following equational axioms on μ -terms to capture bisimulation.

Conway equations

$$\mu x. t[s/x] = t[\mu x. s[t/x]/x]$$

$$\mu x. \mu y. t = \mu x. t[x/y]$$

Group equations

$$(\mu x. t[1 \cdot x/x], \dots, t[n \cdot x/x])_1 = \mu y. (t[y/x_1, \dots, y/x_n])$$

Axioms for branches and bisimulation

$$\begin{array}{lll} s + (t + u) = (s + t) + u & s + t = t + s & t + 0 = t \\ \mu x. x = 0 & \mu x. (x + t) = t & \text{for } t \text{ not containing } x \end{array}$$

The axiom of group equations (Ésik, 2002, 1999b) is a schema read as follows. The notation $(-)_1$ at the left-hand side denotes the first component of a vector. Let n be a natural number and $G = (\{1, \dots, n\}, \cdot)$ be a finite group of order n . Given a vector $x = (x_1, \dots, x_n)$ of distinct variables, define $i \cdot x = (x_{i \cdot 1}, \dots, x_{i \cdot n})$. Thus, $i \cdot x$ is obtained by permuting the components of x according to the i -th row of the multiplication table of G . Varying all possible $n \in \mathbb{N}$ and finite groups G of order n , we generate concrete axioms of "group equations" by using the multiplication \cdot of G . This form is taken from (Ésik, 2002, p.280). Group equations are shown to be an alternative form of the commutative identities. For further details, see (Ésik, 2002), or (Ésik, 1999b, §18 A simple μ -language).

The fixed point law

$$\mu x. t = t[\mu x. t/x]$$

is an instance the first axiom of Conway equations by taking $s = x$.

5.1.2. Equational logic. We call $\text{EL-}\mu$ the standard equational logic of μ -terms with the above axioms. We write $\mu \vdash s = t$ if an equation $s = t$ is derivable from the above axioms in $\text{EL-}\mu$. For example, idempotency is derivable:

$$\mu \vdash t + t = t$$

The proof is $t = \mu x.(x + t) = (\mu x.(x + t)) + t = t + t$, which uses the axiom $\mu x.(x + t) = t$ and the fixed point law. Since μ -terms can be regarded as a representation of process terms of regular behavior (or synchronization trees (Bloom and Ésik, 1993)) as Milner (1984) showed, the standard notion of bisimulation between two μ -terms can be defined (e.g. (Sewell, 1995, page 9)). We write $s \sim t$ if they are bisimilar.

Theorem 5.1. (Bloom et al., 1993; Bloom and Ésik, 1993; Ésik, 2000)((Sewell, 1995, Thm. 2.)) The equational logic $\text{EL-}\mu$ completely axiomatises the bisimulation, i.e. for μ -terms s and t ,

$$\mu \vdash s = t \iff s \sim t.$$

5.2. Characterising UnCAL normal forms

Our strategy to show completeness of our axioms AxG for bisimulation is that we reduce our problem to Thm. 5.1 of the case of μ -terms. Hence, we need to relate UnCAL terms and μ -terms. Clearly, UnCAL terms have richer term constructs than μ -terms. But taking suitable sets of axioms as rewrite rules, we can obtain certain *normal forms*, which are simpler and relate directly to μ -terms. Of course, we should carefully choose axioms from AxG , because some axioms, such as (fix), cause infinite rewrites. We choose the axioms (sub) and (Bekič) from AxG and orient each axiom from left to right as a rewrite rule. Then we have the following rewrite system \mathcal{R} .

$$\begin{aligned} (\text{sub}) \quad t \diamond \langle s_1, \dots, s_n \rangle &\rightarrow t [\vec{y} \mapsto \vec{s}] \\ (\text{Bekič}) \quad \text{cycle}^{X+Y}(\langle t, s \rangle) &\rightarrow \langle \pi_{Z,X}^X, \text{cycle}^Y(s) \rangle \diamond \\ &\quad \langle \text{id}_Z, \text{cycle}^X(t \diamond \langle \text{id}_{Z+X}, \text{cycle}^Y(s) \rangle) \rangle \end{aligned}$$

5.2.1. Termination. The rewrite system \mathcal{R} is terminating, i.e. strongly normalising (SN) (Baader and Nipkow, 1998). We prove it by translating \mathcal{R} to another rewrite system on terms of λG -calculus, which is a simply-typed λ -calculus extended with fixed point operator given in §6.3. (The λG -calculus does not depend on the result of completeness for bisimulation we establish in this section, hence it is not a circular argument.) We use the translation $\llbracket - \rrbracket$ from UnCAL terms to λG -terms given in Fig. 12 in Appendix. Applying this translation to \mathcal{R} , we obtain the rewrite rules \mathcal{R}_G on λG -terms:

$$\begin{aligned} (\text{sub}') \quad (\lambda \vec{y}.t) (s_1, \dots, s_n) &\rightarrow t [\vec{y} \mapsto \vec{s}] \\ (\text{Bekič}') \quad \text{fix}_{m+n}(\lambda(\vec{x}, \vec{y}).(\hat{t}, \hat{s})) &\rightarrow (\text{fix}_m(\lambda \vec{x}.(\lambda \vec{y}.\hat{t}) \text{fix}_n(\lambda \vec{y}.\hat{s})), \\ &\quad \text{fix}_n(\lambda \vec{y}.(\lambda \vec{x}.\hat{s}) \text{fix}_m(\lambda \vec{x}.(\lambda \vec{y}.\hat{t}) \text{fix}_n(\lambda \vec{y}.\hat{s})))) \end{aligned}$$

where \hat{t} and \hat{s} are short for $t(\vec{x}, \vec{y})$ and $s(\vec{x}, \vec{y})$, respectively.

The rewrite system \mathcal{R}_G is SN. Basic reasons are as follows. The rule (sub') is essentially the β -reduction rule of the simply-typed λ -calculus, hence terminating. Applying (Bekič') rule, the number of the tuples at the argument of fix is reduced. Namely, the subscript of fix (i.e. m or n) in the right-hand side of (Bekič') is always smaller than that in the left-hand side (i.e. $m + n$), hence the rule (Bekič') is terminating.

More formally, we can prove SN of \mathcal{R}_G using a general established method called the General Schema (Blanqui et al., 2002; Blanqui, 2000), which is based on Tait's computability method to show SN. The General Schema has succeeded to prove termination of various recursors such as the recursor in Gödel's System T. The basic idea of the General Schema is to check whether the arguments of recursive calls in the the right-hand side of a rewrite rule are "smaller" than the left-hand sides' ones. It is similar to Coquand's notion of "structurally smaller" (Coquand, 1992), but more relaxed and extended. The rewrite system \mathcal{R}_G can be seen as a rewrite system of the format of inductive datatype system given in (Blanqui, 2000). Note that s 's and t are metavariables in an inductive datatype system, and the abstraction $\lambda x.t$ is written as $[x]t$ in (Blanqui, 2000). Since \mathcal{R}_G fits into the the General Schema using the well-founded order $\text{cycle}^m > \text{cycle}^n$ where natural numbers $m > n$, \mathcal{R}_G is SN. A larger rewrite system involving these rules has been proved to be SN by using the General Schema in (Hamana, 2016).

It is straightforward to show that if we have $s \rightarrow_{\mathcal{R}} t$ using \mathcal{R} , then we have $\llbracket s \rrbracket \rightarrow_{\mathcal{R}_G} \llbracket t \rrbracket$ using \mathcal{R}_G . Suppose that there is an infinite rewrite sequence:

$$s_0 \rightarrow_{\mathcal{R}} s_1 \rightarrow_{\mathcal{R}} s_2 \rightarrow_{\mathcal{R}} \cdots$$

It is translated to an infinite rewrite sequence in the λG -calculus:

$$\llbracket s_0 \rrbracket \rightarrow_{\mathcal{R}_G} \llbracket s_1 \rrbracket \rightarrow_{\mathcal{R}_G} \llbracket s_2 \rrbracket \rightarrow_{\mathcal{R}_G} \cdots$$

But this is impossible because \mathcal{R}_G is SN. So \mathcal{R} is SN.

5.2.2. Confluence. The rewrite system \mathcal{R} is confluent because it is left-linear (i.e. there is no repetition of the same metavariable at every left-hand side) and there is no overlapping between the rules (Klop, 1980; Klop et al., 1993).

5.2.3. Unique normal forms. Since \mathcal{R} is confluent and terminating, any term has the unique normal form by rewriting using \mathcal{R} (Baader and Nipkow, 1998). Given an UnCAL term, we write

$$\text{nf}(t)$$

for the unique normal form of t .

Lemma 5.2. If t is of type $X + Y$ where $n = |X + Y| > 1$, then $\text{nf}(t)$ is of the form $\langle t_1, \dots, t_n \rangle$.

Proof. A possible term t of such a type is either $\langle t_1, \dots, t_n \rangle$ or $\text{cycle}^{X+Y}(\langle s, t \rangle)$. For the latter case, applying (Bekič), $\text{cycle}^{X+Y}(\langle s, t \rangle)$ becomes a tuple involving $\text{cycle}^X(s), \text{cycle}^Y(t)$ and \diamond . The \diamond is consumed by (sub), and applying (Bekič) repeatedly the subterms $\text{cycle}^X(s), \text{cycle}^Y(t)$ finally become cycle^1 -terms. Hence the normal form of t is a tuple. \square

We call a raw term t *value* if it follows the following grammar

$$s, t ::= y \quad | \quad \ell : t \quad | \quad \text{cycle}^x(x \triangleleft t) \quad | \quad \{\} \quad | \quad \lambda \quad | \quad \lambda \diamond \langle s, t \rangle \quad | \quad (x \triangleleft t) \\ | \quad \langle s, t \rangle \quad | \quad ()$$

Note that variables x, y are arbitrary, which may be the same markers or may be different.

Lemma 5.3. We define $\mathcal{M} \triangleq \{t \mid Y \vdash t : X \text{ and } t \text{ is a value}\}$. For any $Y \vdash t : X$, we have $\text{nf}(t) \in \mathcal{M}$.

Proof. By induction on typing derivations of t .

- Case (Emp)(Pair),(Nil),(Man),(Mark). It is in \mathcal{M} .
- Case (Def) $t = (x \triangleleft s)$. Let $s' = \text{nf}(s)$. By I.H., $s' \in \mathcal{M}$. Hence $(x \triangleleft s') \in \mathcal{M}$.
- Case (Label) Similar to (Def).
- Case (Com) $t = u \diamond s$.
 - Case $t = \lambda \diamond s$. Then $\text{nf}(s) = \langle s_1, s_2 \rangle$ and s_1, s_2 are normal forms. By I.H., $s_1, s_2 \in \mathcal{M}$. Then $\text{nf}(t) = \lambda \diamond \langle s_1, s_2 \rangle \in \mathcal{M}$.
 - Case $u \neq \lambda$. Let u', s' be the normal forms of u, s . By I.H., these are in \mathcal{M} . By Lemma 5.2, $s' = \langle s'_1, \dots, s'_n \rangle$ (n is possibly 0, 1). So $\text{nf}(u' \diamond \langle s'_1, \dots, s'_n \rangle) = u' [\vec{x} \mapsto s'_1, \dots, s'_n] \in \mathcal{M}$.
- Case (Cyc) $t = \text{cycle}^X(s)$ with $n = |X|$.
 - $n > 1$. Then $\text{nf}(s) = \langle s_1, \dots, s_n \rangle$ and every $s_i \in \mathcal{M}$. By Lemma 5.2, $\text{nf}(t) = \text{nf}(\text{cycle}^X(\langle s_1, \dots, s_n \rangle))$ is a tuple consisting of s_i , hence in \mathcal{M} .
 - $n = 1$. Similar to (Def). □

Proposition 5.4. If $t \in \mathcal{M}$ is of type $\&$, then t follows the grammar

$$\mathcal{M}_{\&} \ni s, t ::= y \quad | \quad \ell : t \quad | \quad \text{cycle}^x(x \triangleleft t) \quad | \quad \{\} \quad | \quad \lambda \quad | \quad \lambda \diamond \langle s, t \rangle$$

Proof. Clear from the type. □

We regard $\mathcal{M}_{\&}$ as a term set. As the final normalisation process, we generate another term set \mathcal{N} from $\mathcal{M}_{\&}$ by replacing every sole $\lambda_{\langle y_1, y_2 \rangle}$ (which is not of the form $\lambda_{\langle y_1, y_2 \rangle} \diamond \langle s, t \rangle$) in terms of $\mathcal{M}_{\&}$, with the “ η -expansion” $\lambda_{\langle y_1, y_2 \rangle} \diamond \langle y_1, y_2 \rangle$. This process uses the axiom $(\eta \lambda)$ in AxG. Thus, we have

$$\begin{array}{lcl} \mathcal{N} \ni & s, t ::= & y \quad | \quad \ell : t \quad | \quad \text{cycle}^x(x \triangleleft t) \quad | \quad \{\} \quad | \quad \lambda \diamond \langle s, t \rangle \\ \mathcal{T}_{\mu}(V) \ni & s, t ::= & y \quad | \quad \ell(t) \quad | \quad \mu x. t \quad | \quad 0 \quad | \quad s + t \end{array}$$

We call an element of \mathcal{N} *UnCAL normal form*, which is always of type $\&$. Every UnCAL normal form bijectively corresponds to a μ -term in $\mathcal{T}_{\mu}(V)$, i.e. $\mathcal{N} \cong \mathcal{T}_{\mu}(V)$, because each the above construct corresponds to the lower one. Hereafter, we may identify normal forms in \mathcal{N} and μ -terms as above.

Definition 5.5. We say that well-typed UnCAL terms s and t are *bisimilar*, denoted by $s \sim t$, if the UnCAL graphs $\llbracket s \rrbracket_{\mathbf{Gr}}^{\{\&\}}$ and $\llbracket t \rrbracket_{\mathbf{Gr}}^{\{\&\}}$ are extended bisimilar described in Appendix §A.1.

For a μ -term t , its *chart* (Sewell, 1995, page 9) is extended bisimilar to the UnCAL graph $\llbracket t \rrbracket_{\mathbf{Gr}}^{\{\&\}}$. This is shown by induction on the term syntax. Then the bisimilarity for μ -terms agrees with the bisimilarity for UnCAL normal forms in \mathcal{N} .

5.3. Completeness of the axioms for bisimulation

By Thm. 5.1, we have known that EL- μ is complete for bisimulation. We show the completeness of AxG in EL-UnCAL for bisimulation, using the following Lemma 5.6 that relates the problem of EL- μ with that EL-UnCAL of through UnCAL normal forms.

Lemma 5.6. For UnCAL normal forms $n, m \in \mathcal{N}$, if $\mu \vdash n = m$ then there exists a marker x such that

$$Y \vdash n = m : \langle\langle x \rangle\rangle$$

is derivable from AxG in EL-UnCAL. Note that x is mostly $\&$, but consider the case $n = \text{cycle}^x(t)$.

Proof. By induction on proofs of EL- μ . For every axiom in EL- μ , there exists a corresponding axiom in AxG or an EL-UnCAL theorem, hence it can be emulated in EL-UnCAL. \square

Theorem 5.7. (Completeness) Assume a pure UnCAL theory. For any terms s and t of type x ,

$$Y \vdash s = t : \langle\langle x \rangle\rangle \text{ is derivable from AxG in EL-UnCAL} \quad \text{iff} \quad s \sim t.$$

Proof. $[\Rightarrow]$: Because for every axiom $s = t$ in AxG, we have $s \sim t$, and the bisimulation is closed under contexts and substitutions (Buneman et al., 2000).

$[\Leftarrow]$: Suppose $s \sim t$. Since for every rewrite rule in \mathcal{R} , both sides of the rule is bisimilar, nf preserves the bisimilarity. So we have $\text{nf}(s) \sim s \sim t \sim \text{nf}(t)$. Since normal forms correspond to μ -terms, using Thm. 5.1, we have $\mu \vdash \text{nf}(s) = \text{nf}(t)$. By Lemma 5.6, we have a theorem $Y \vdash \text{nf}(s) = \text{nf}(t) : \langle\langle x \rangle\rangle$. Thus $s = t$ is derivable. \square

6. Instances of UnCAL Models

The categorical semantics provides a *generic framework* of semantics of UnCAL. It is not only providing a single semantics of UnCAL, but also captures various *concrete models* of UnCAL by the general form of categorical semantics, because it is *parameterised* by an *arbitrary* iteration category \mathcal{C} . To give a concrete model, we *choose* a concrete iteration category \mathcal{C} with a concrete L -monoid in it. We have given the syntactic model \mathbf{U} in \mathcal{Cl}_E in §3.2. In this section, we will give several other concrete models.

6.1. The bisimulation model

Buneman et al. formulated that UnCAL graphs were identified by a sort of bisimulation called extended bisimulation (Buneman et al., 2000, Definition 3). We give this bisimulation model as an example of UnCAL model. We define an equivalence relation \sim_E on well-typed terms by the symmetric transitive closure of the union of the renaming identification, the bisimulation \sim on UnCAL terms given in Def. 5.5 and the congruence generated by the additional axioms. Crucially, our equational axiomatisation in EL-UnCAL is complete for bisimulation.

Theorem 6.1. (Completeness for bisimulation) Let E be UnCAL axioms (including AxG). Then $Y \vdash s = t : X$ is derivable in EL-UnCAL iff $s \sim_E t$.

Proof. Immediate corollary to Thm. 5.7. \square

Thus, the equivalence relation generated by E in the equational logic EL-UnCAL is exactly the same as bisimulation with additional axioms. Hence the generic model U in the classifying category $\mathcal{C}\ell_E$ considered in §3.2 is also the terms modulo bisimulation model.

The original graph theoretic definition of UnCAL graphs, extended bisimilarity, and constructors form also an UnCAL model, which is described in detail in Appendix A.

6.2. A CPO model

We next give a new model of UnCAL. One of the most natural examples of iteration category is the category **CPO** of cpos (complete partial orders) having the least element \perp and continuous functions. It is well-known that the category **CPO** is traced cartesian (Hasegawa, 1997a), moreover an iteration category (Bloom and Ésik, 1993), where the least fixed point operator (calculated by $\bigsqcup_{n \in \mathbb{N}} f^n(\perp)$ for a continuous function f) is a Conway operator $(-)^{\dagger}$. In the category **CPO**, a cycle is interpreted as a suitable infinite unfolding.

Let L^ω denote the set of all finite and infinite strings of labels L . We define D to be the set of all non-empty subsets of L^ω modulo the equivalence relation generated by the preorder \sqsubseteq , where $A \sqsubseteq B$ iff for all $u \in A$, there exists $w \in B$ such that u is a prefix of w . This D with the order \sqsubseteq (extended to the quotient sets) forms a cpo, actually a Hoare powerdomain $\mathcal{P}(L^\omega)$ (Abramsky and Jung, 1994). The least element \perp is the singleton $\{\varepsilon\}$ of the empty string. We now define an L -monoid structure on D by taking $\eta^D \triangleq \perp = \{\varepsilon\}$, μ^D as the binary join \bigsqcup , and $\ell^D(A) \triangleq \{\ell w \mid w \in A\}$ for $\ell \in L$.

Then D satisfies all the axioms in AxG (see the examples below). The categorical interpretation $\llbracket - \rrbracket_{\mathbf{CPO}}^D$, induced from Fig. 10, translates an UnCAL term $Y \vdash t : X$ to a continuous function

$$\llbracket t \rrbracket_{\mathbf{CPO}}^D : D^{|Y|} \rightarrow D^{|X|}.$$

Example 6.2. Since D is a model, any UnCAL theorem $s = t$ is valid using D , i.e. $\llbracket s \rrbracket_{\mathbf{CPO}}^D = \llbracket t \rrbracket_{\mathbf{CPO}}^D$ holds. We demonstrate the interpretation of theorems (except for (5)).

$$(1) \llbracket (\mathbf{a}:x) \diamond (x \triangleleft \mathbf{b}: \{\}) \rrbracket = (\lambda x. \mathbf{a}^D(x)) \{\mathbf{b}\} = \{\mathbf{ab}\} = \llbracket \mathbf{a}:\mathbf{b}: \{\} \rrbracket$$

- (2) $\llbracket \{\mathbf{a}: \{\}\} \cup \{\} \cup \{\mathbf{b}: \mathbf{c}: \{\}\} \cup \{\mathbf{a}: \{\}\} \rrbracket = \{\mathbf{a}, \mathbf{bc}\} = \llbracket \{\mathbf{a}: \{\}\} \cup \{\mathbf{b}: \mathbf{c}: \{\}\} \rrbracket$
- (3) $\llbracket \text{cycle}(\&) \rrbracket = (\lambda x. x)^\dagger = \perp = \{\varepsilon\} = \llbracket \{\} \rrbracket$,
- (4) $\llbracket y \vdash \text{cycle}^\&(\lambda \langle \&, y \rangle) : \& \rrbracket = \lambda y. (\lambda x. \mu^D(x, y))^\dagger = \lambda y. \bigsqcup_{n \in \mathbb{N}} ((\lambda x. \mu^D(x, y))^n)(\perp) = \lambda y. y = \llbracket y \vdash y : \& \rrbracket$
- (5) $\llbracket \text{cycle}(\mathbf{a}: \&) \rrbracket = (\lambda x. \mathbf{a}^D(x))^\dagger = \bigsqcup_{n \in \mathbb{N}} (\lambda x. \mathbf{a}^D(x))^n(\perp) = \bigsqcup \{\{\mathbf{a}^n\} \mid n \in \mathbb{N}\} = \{\mathbf{a}^\omega\}$

There are several benefits of this model.

- (i) The cpo model explains in a mathematically rigorous way how the infinite expansion of a cyclic graph occurs, i.e. it is the least fixed point calculated by the least upper bound of an ω -chain (see (4) and (5)).
- (ii) The cpo model gives *intuition* why the axioms hold. For example, “ $\{\}$ ” can be understood as \perp , and a reason why the axioms (c1) and (c2) hold can be understood by the fixed points (see (3) and (4)) and the structure of cpos.
- (iii) This *motivates* the following idea: a λ -calculus with fixed point operator may be used to calculate values and to check equality (e.g. (1)) of UnCAL terms. We will realise this idea in the following subsection.

6.3. The λG -calculus model

UnCAL is very similar to an ordinary functional programming language, in the sense that UnCAL programs are given by structural recursive definitions. Also, in the examples (1)–(5) and the item (iii) in §6.2, we observed some similarities between UnCAL and a λ -calculus.

But UnCAL’s target data was graphs (rather than data of algebraic data types) and the computation of UnCAL was realised by graph algorithms. Thus it has been considered that UnCAL is far from an ordinary functional programming language. But there must be some connection to functional programming. What is the exact connection?

Using our categorical semantics, we can now formally answer to the question. In this subsection, we show a connection between UnCAL and functional programming by giving an interesting translation from UnCAL terms to λ -terms. Crucially, this translation is *automatically obtained as an instance of our categorical semantics*. That is, we first define an applied simply-typed λ -calculus called the λG -calculus, and next show that it forms an UnCAL model. Then it automatically induces a categorical interpretation, which gives a sound and complete translation from UnCAL terms and programs to λ -terms in the λG -calculus.

6.3.1. The λG -calculus. Suppose a label set L is given. The λG -calculus is a simply-typed lambda calculus extended with pairs, constants, fixed point operator with the axioms in Fig. 11 and possibly additional axioms Ax. We assume three base types, \mathbf{B} (for Booleans), \mathbf{G} (for graphs), and \mathbf{L} (for labels). We identify $\mathbf{1} \times \tau = \tau \times \mathbf{1} = \tau$. We also identify $\tau_1 \times (\tau_2 \times \tau_3) = (\tau_1 \times \tau_2) \times \tau_3$, simply write $\tau_1 \times \tau_2 \times \tau_3$ and corresponding elements as (t_1, t_2, t_3) , and assume that π_i is suitably defined for n -case. We use the

notation $\lambda(x_1, \dots, x_n).t$ (or $\lambda \vec{x}.t$) to denote $\lambda x^{\sigma_1 \times \dots \times \sigma_n}.t [\pi_1 x_1, \dots, \pi_n x / x_1, \dots, x_n]$. We also identify a curried term with its uncurried one, i.e., by $\lambda(x_1, \dots, x_n).\lambda(y_1, \dots, y_m).t$, we mean $\lambda(x_1, \dots, x_n, y_1, \dots, y_m).t$. This is because in our use of λ G-calculus, we are mainly interested in up to first-order types.

The typing rules are exactly the standard rules for the simply-typed λ -calculus (e.g. (Crole, 1993)) using the typed constants given below. Note that the constant \bullet corresponds to $\{\}$ in UnCAL, semantically corresponds to \perp in the cpo semantics. It is a formalisation of the “black hole” (“apparent undefinedness”) used in a call-by-need calculus by Nakata and Hasegawa (2009).

As usual, every axiom (including Ax) consists of a pair of well-typed terms of the same type and under the same context. Most of the axioms for λ G are obtained by translating the axioms AxG in EL-UnCAL using the translation $\llbracket - \rrbracket$ in Fig. 12 in Appendix. The axioms of parameterised fixed point are slightly different, but they are a version of the axioms of Conway parameterised fixed point μ -terms (Ésik, 2002). (amalg) is the axiom schema called *amalgamation* (Simpson and Plotkin, 2000, Def. 3.2) or *functorial implication* (Bloom and Ésik, 1994, Def. 8), which is very close to the commutative identities (CI) in AxG. Note that (c1) is derivable as in AxG (but we include it for readability). The λ G-theory is an equational theory obtained by ordinary equational logic of λ -calculus using the axioms in Fig. 11 and additional axioms Ax.

6.3.2. Categorical semantics. We define the category **Lam** of first-order λ G-terms by taking

- *objects*: types G^n ($n \in \mathbb{N}$)
- *arrows*: arrows from G^m to G^n are equivalence classes $[\vdash t : G^m \rightarrow G^n]$ of closed terms quotiented by the congruence generated by renaming, the axioms of λ G and additional axioms Ax
- *composition*: $[\vdash \lambda y.s : G^k \rightarrow G^n] \circ [\vdash \lambda x.t : G^m \rightarrow G^k] = [\vdash \lambda x.s[t/y] : G^m \rightarrow G^n]$
- *identity*: $[\vdash \lambda x.x : G^n \rightarrow G^n]$

Then, because of the axioms, it is immediate that **Lam** forms an iteration category, where the Conway operator $(-)^{\dagger}$ is given by fix . Moreover, G forms an L -monoid by taking

$$[\bullet] : \mathbf{1} \rightarrow G, \quad [\cup] : G \times G \rightarrow G, \quad \llbracket \ell \rrbracket_L^G = [\lambda x.\ell : x].$$

The categorical interpretation in Fig. 10 induces the interpretation $\llbracket - \rrbracket_{\mathbf{Lam}}^G$ that interprets an UnCAL term $Y \vdash t : X$ as a closed λ -term of first-order type:

$$\llbracket t \rrbracket_{\mathbf{Lam}}^G : G^{|Y|} \rightarrow G^{|X|}.$$

The interpretation $\llbracket - \rrbracket_{\mathbf{Lam}}^G$ is concretely described as the interpretation function $\llbracket - \rrbracket$ in Fig. 12 in Appendix. The examples considered in Example 6.2 are now interpreted as follows.

$$(1) \llbracket (\mathbf{a}:x) \diamond (x \triangleleft \mathbf{b}:\{\}) \rrbracket = [(\lambda x.\mathbf{a}:x) (\mathbf{b}:\bullet)] = [\mathbf{a}:\mathbf{b}:\bullet]$$

The λG -calculus

<i>Types</i>	$\tau ::= \mathbf{1} \mid \mathbf{G} \mid \mathbf{L} \mid \mathbf{B} \mid \tau \times \tau' \mid \tau \rightarrow \tau'$
<i>Terms</i>	$e ::= x \mid \lambda x.e \mid e_1 e_2 \mid (e_1, e_2) \mid c \quad (\text{constant})$
<i>Constants</i>	$() : \mathbf{1} \quad \pi_i : \tau_1 \times \tau_2 \rightarrow \tau_i \text{ for } i = 1, 2 \quad \bullet : \mathbf{G} \quad \cup : \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$ $- : - : \mathbf{L} \times \mathbf{G} \rightarrow \mathbf{G} \quad \text{fix}_n : (\mathbf{G}^n \rightarrow \mathbf{G}^n) \rightarrow \mathbf{G}^n \quad \ell : \mathbf{L} \text{ for } \ell \in L$ $\text{true} : \mathbf{B} \quad \text{false} : \mathbf{B} \quad \text{if } - \text{ then } - \text{ else } - : \mathbf{B} \times \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G} \quad - \equiv - : \mathbf{L} \times \mathbf{L} \rightarrow \mathbf{L}$

Axioms

Calculus

(beta)	$(\lambda x.t) s = t[s/x]$	(Cw1)	$\text{fix}(\lambda x.t[s/x]) = t[\text{fix}(\lambda x.s[t/x])/x]$
(eta)	$(\lambda x.tx) = t$	(Cwn2)	$\text{fix}(\lambda x.\text{fix}(\lambda y.t)) = \text{fix}(\lambda x.t[x/y])$

Cartesian

(fst)	$\pi_1(s, t) = s$	(amalg)	$\text{fix}(\lambda(x_1, \dots, x_n).(t_1, \dots, t_n))$ $= (\text{fix}(\lambda y.s), \dots, \text{fix}(\lambda y.s))$ if $t_i[y, \dots, y/x_1, \dots, x_n] = s$ for all t_i
(snd)	$\pi_2(s, t) = t$		
(fsi)	$(\pi_1 t, \pi_2 t) = t$		

Deleting trivial cycle

(c1)	$\text{fix}(\lambda x.x) = \bullet$	(unitL \cup)	$\bullet \cup t = t$
\star (c2)	$\text{fix}(\lambda x.(x \cup t)) = t$	(unitR \cup)	$t \cup \bullet = t$
		(assoc \cup)	$s \cup (t \cup u) = (s \cup t) \cup u$
		\star (com \cup)	$s \cup t = t \cup s$
		\star (degen)	$t \cup t = t$

Conditionals

(ift)	if true then t else $e = t$
(ife)	if false then t else $e = e$
(eqt)	$\ell \equiv \ell = \text{true}$ for $\ell \in L$
(eqe)	$\ell \neq \ell' = \text{false}$ for $\ell \neq \ell' \in L$

Note. We use the abbreviation $s \cup t \triangleq \cup(s, t)$. In (eta) and (c2), x does not appear in t . The subscript $n \in \mathbb{N}$ of fix may be omitted for simplicity.

Fig. 11. The λG -calculus

$$(2) \llbracket \{\mathbf{a} : \{\}\} \cup \{\} \cup \{\mathbf{b} : \mathbf{c} : \{\}\} \cup \{\mathbf{a} : \{\}\} \rrbracket = [\mathbf{a} : \bullet \cup \bullet \cup (\mathbf{b} : \mathbf{c} : \bullet) \cup (\mathbf{a} : \bullet)] = [(\mathbf{a} : \bullet) \cup (\mathbf{b} : \mathbf{c} : \bullet)]$$

$$(3) \llbracket \text{cycle}(\&) \rrbracket = [\text{fix}(\lambda x.x)] = [\bullet]$$

$$(4) \llbracket y \vdash \text{cycle}^{\&}(\lambda \langle \&, y \rangle) : \& \rrbracket = [\lambda y. \text{fix}(\lambda x.x \cup y)] = [\lambda y. y]$$

$$(5) \llbracket \text{cycle}(\mathbf{a} : \&) \rrbracket = [\text{fix}(\lambda x.\mathbf{a} : x)] = [\mathbf{a} : \text{fix}(\lambda x.\mathbf{a} : x)] = [\mathbf{a} : \mathbf{a} : \text{fix}(\lambda x.\mathbf{a} : x)]$$

$$(6) \llbracket y_1, y_2 \vdash \mathbf{t}_G : \& \rrbracket = \llbracket \mathbf{a} : (\{\mathbf{b} : x\} \cup \{\mathbf{c} : x\}) \diamond \text{cycle}(x \triangleleft \mathbf{d} : (\{\mathbf{p} : y_1\} \cup \{\mathbf{q} : y_2\} \cup \{\mathbf{r} : x\})) \rrbracket \\ = [\lambda(y_1, y_2). (\lambda x.\mathbf{a} : (\mathbf{b} : x) \cup (\mathbf{c} : x)) \text{fix}(\lambda x.\mathbf{d} : ((\mathbf{p} : y_1) \cup (\mathbf{q} : y_2) \cup (\mathbf{r} : x)))]$$

The last two examples correspond to the examples considered in Introduction, §1.3.1.

Proposition 6.3. Given UnCAL's additional axioms \mathcal{E} , we define λG 's additional axioms \mathcal{E}' by $\mathcal{E}' \triangleq \{\llbracket s \rrbracket = \llbracket t \rrbracket \mid (s = t) \in \mathcal{E}\}$. Suppose $Y \vdash s : X$ and $Y \vdash t : X$. Then

$Y \vdash s = t : X$ is derivable in EL-UnCAL iff $\llbracket s \rrbracket_{\mathbf{Lam}}^G = \llbracket t \rrbracket_{\mathbf{Lam}}^G$ holds in the λG -theory.

Proof. $[\Rightarrow]$: By soundness of the categorical semantics.

$[\Leftarrow]$: We define the inverse translation $(-)^{\circ}$ of $\llbracket - \rrbracket_{\mathbf{Lam}}$, i.e. a mapping from the image of $\llbracket - \rrbracket_{\mathbf{Lam}}^G$ to $\mathcal{Cl}_E(m, n)$ by

$$\begin{aligned} [\lambda \vec{y}.y_i]^{\circ} &= [i]_E & [\lambda y.s \ t]^{\circ} &= [\lambda y.s]^{\circ} \circ \langle \text{id}_{U^m}, [\lambda y.t]^{\circ} \rangle \\ [\lambda y.(s, t)]^{\circ} &= \langle [\lambda y.s]^{\circ}, [\lambda y.t]^{\circ} \rangle & [\lambda y.\text{fix}(t)]^{\circ} &= ([\lambda y.t]^{\circ})^{\dagger} \\ [\lambda y.\ell : t]^{\circ} &= \llbracket \ell \rrbracket_L \circ [\lambda y.t]^{\circ} & [\lambda y.()]^{\circ} &= [()]_E \\ [\pi_i]^{\circ} &= [\pi_i]_E & [\lambda y.\bullet]^{\circ} &= [\{\}]_E \quad [\cup]^{\circ} = [\wedge]_E \end{aligned}$$

Note that for $[t]^{\circ}$, any subterm of t , other than fix , is up to first-order type. It defines a partial function

$$(-)^{\circ} : \mathbf{Lam}(\mathbf{G}^m, \mathbf{G}^n) \rightarrow \mathcal{Cl}_E(m, n).$$

which is well-defined, i.e., for closed terms $t, t' : \mathbf{G}^m \rightarrow \mathbf{G}^n$, where any subterm of t, t' , other than fix , is up to first-order type, the following holds:

if $t = t'$ holds in the λG -theory, then $t^{\circ} = t'^{\circ}$ is an UnCAL theorem.

This is proved by induction on the proof of $t = t'$. We need to check that for every axiom $s = s'$ of the λG -calculus, $s^{\circ} = s'^{\circ}$ is an UnCAL theorem. By applying $(-)^{\circ}$ to an axiom, we can recover the corresponding axiom in AxG or obtain an UnCAL theorem. Note that the axioms for λG were obtained by translating each of axiom AxG to that of λG using $\llbracket - \rrbracket_{\mathbf{Lam}}^G$. Then we have $(\llbracket t \rrbracket_{\mathbf{Lam}}^G)^{\circ} = [t]_E$ for all UnCAL terms t , by induction on typing derivations. Suppose $\llbracket s \rrbracket_{\mathbf{Lam}}^G = \llbracket t \rrbracket_{\mathbf{Lam}}^G$. Applying $(-)^{\circ}$, we obtain $[s]_E = (\llbracket s \rrbracket_{\mathbf{Lam}}^G)^{\circ} = (\llbracket t \rrbracket_{\mathbf{Lam}}^G)^{\circ} = [t]_E$. \square

Hence any provable equation $s = t$ over UnCAL terms in EL-UnCAL is also provable using λG -calculus. If we have suitable operational semantics for λG -calculus, we may prove it by simplifying terms by the operational semantics in λG . This methodology is especially needed when one wants to prove the form $s = v$ where v is a suitable “value” in UnCAL. It is actually possible via a functional programming language called FUnCAL, which we will see in §6.3.5.

6.3.3. Axiomatisation of the recursion operators. To translate UnCAL programs defined by structural recursion to λG , next we axiomatise the recursion operators formally in the λG -calculus. We assume additionally new constants **srec** for structural recursion, **prec** for primitive recursion, and the following axioms in λG .

Axioms for recursion operators

$$\begin{aligned}
\mathbf{srec} &: (\mathbf{L} \rightarrow \mathbf{G}^k \rightarrow \mathbf{G}^k) \rightarrow (\mathbf{G}^n \rightarrow \mathbf{G}^m) \rightarrow (\mathbf{G}^{k \cdot n} \rightarrow \mathbf{G}^{k \cdot m}) \\
\mathbf{srec} \, e \, (\lambda \overrightarrow{y}. y_i) &= \pi'_i \\
\mathbf{srec} \, e \, (\lambda y. s \, t) &= \lambda y'. (\mathbf{srec} \, e \, \lambda y. s) \, (y', (\mathbf{srec} \, e \, \lambda y. t) \, y') \\
\mathbf{srec} \, e \, (\lambda y. (s, t)) &= \mathbf{srec} \, e \, (\lambda y. s) \times' \mathbf{srec} \, e \, (\lambda y. t) \\
\mathbf{srec} \, e \, (\lambda y. \text{fix}(t)) &= \lambda y'. \text{fix}((\mathbf{srec} \, e \, \lambda y. t) \, y') \\
\mathbf{srec} \, e \, (\lambda y. \ell : t) &= \lambda y'. e \, \ell \, ((\mathbf{srec} \, e \, \lambda y. t) \, y') \\
\mathbf{srec} \, e \, (\lambda y. ()) &= \lambda y'. () \\
\mathbf{srec} \, e \, (\pi_i) &= \pi'_i \\
\mathbf{srec} \, e \, (\lambda y. \bullet) &= \lambda y'. (\bullet, \dots, \bullet) \\
\mathbf{srec} \, e \, (\cup) &= \cup'
\end{aligned}$$

$$\begin{aligned}
\mathbf{prec} &: (\mathbf{L} \rightarrow \mathbf{G}^{k+1} \rightarrow \mathbf{G}^k) \rightarrow \mathbf{G}^m \rightarrow \mathbf{G}^{k \cdot m} \\
\mathbf{prec} \, e \, t &= \pi \, (\mathbf{srec} \, (\lambda \ell \lambda x. (e \, \ell \, x, \ell : \pi' x)) \, t)
\end{aligned}$$

The types of the recursion operators \mathbf{srec} and \mathbf{prec} correspond to those of functions Ψ in §4.2 and Υ in §4.3, respectively, which are actually parameterised by $k, m, n \in \mathbb{N}$ (but we omit to attach subscripts).

The operator π' selects the final coordinate of a tuple x , π selects the rest of it, and π'_i is the i -th projection of a tuple of tuples. Moreover, \times' is the concatenating operator of two tuples under λ -binder, \cup' is the “zip” operator of two tuples by \cup under λ -binder. Here, by an operator, we mean a suitable λ -term.

We call this extension with additional axioms translated from UnCAL’s additional axioms *the extended λG -calculus* and its theory *the extended λG -theory*. These axioms are a formalisation of the characterisation we have obtained in (3).

6.3.4. Relating UnCAL and λG . Suppose the structural recursion $\mathbf{srec}(e)$ for the L -indexed expressions $e = (e_\ell)_{\ell \in L}$ of UnCAL, where $W \vdash e_\ell : W$ is given. We assume that e for $\mathbf{srec}(e)$ must satisfy the condition that there exists a λG -term $\llbracket e \rrbracket : \mathbf{L} \rightarrow \mathbf{G}^{|W|} \rightarrow \mathbf{G}^{|W|}$ such that

$$\llbracket e \rrbracket \, \ell = \llbracket e_\ell \rrbracket_{\mathbf{Lam}}^{\mathbf{G}}$$

holds in the extended λG -theory. This assumption means that a structural recursive definition is given by a suitable case analysis. For example, the function $\mathbf{aa?}$ in Example 4.1 is represented as $\mathbf{srec}(e)$ where $e_a = \text{head-a?}(t)$ and $e_\ell = \mathbf{aa?}(t)$ for $\ell \neq a$. In this case, we can take

$$\llbracket e \rrbracket \triangleq \lambda \ell t. \text{if } \ell \equiv a \text{ then head-a?}(t) \text{ else aa?}(t)$$

Similarly for $\mathbf{prec}(e)$, we assume that for $W, \& \vdash e : W$, there exists a λG -term $\llbracket e \rrbracket : \mathbf{L} \rightarrow \mathbf{G}^{|W|+1} \rightarrow \mathbf{G}^{|W|}$ such that $\llbracket e \rrbracket \, \ell = \llbracket e_\ell \rrbracket_{\mathbf{Lam}}^{\mathbf{G}}$ holds in the extended λG -theory.

The axiomatisation is sound and complete as follows.

Proposition 6.4. For any term $Y \vdash t : X$,

$\mathbf{srec}(e)(t) = u$ iff $\mathbf{srec}(\lambda\ell. \llbracket e \rrbracket \ell) (\llbracket t \rrbracket_{\mathbf{Lam}}) = \llbracket u \rrbracket_{\mathbf{Lam}}$ in the extended $\lambda\mathbf{G}$ -theory,
 $\mathbf{prec}(e)(t) = u$ iff $\mathbf{prec}(\lambda\ell. \llbracket e \rrbracket \ell) (\llbracket t \rrbracket_{\mathbf{Lam}}) = \llbracket u \rrbracket_{\mathbf{Lam}}$ in the extended $\lambda\mathbf{G}$ -theory.

Proof. $[\Rightarrow]$: By induction on the typing derivation of t .

$[\Leftarrow]$: As in Prop. 6.3, we define the inverse translation also for \mathbf{srec} and \mathbf{prec} .

$$[\mathbf{srec}(\lambda\ell. \llbracket e \rrbracket \ell) t]^\circ \triangleq \mathbf{srec}(e)([t]^\circ)$$

$$[\mathbf{prec}(\lambda\ell. \llbracket e \rrbracket \ell) t]^\circ \triangleq \mathbf{prec}(e)([t]^\circ)$$

The rest is similar to Prop. 6.3. □

These theorems open possibilities of importing the techniques and theory developed for the λ -calculus and functional programming to UnCAL. It enhances reasoning, computations and optimisations of UnCAL programs. One can translate an equational problem of UnCAL into the extended $\lambda\mathbf{G}$ -calculus, then one can use reasoning and computation in $\lambda\mathbf{G}$ and retrieve the result by translating back it to UnCAL by $[-]^\circ$ given in the proof of Prop. 6.3.

6.3.5. The language FUnCAL as an efficient operational semantics of $\lambda\mathbf{G}$. Finally, we mention that our clarification of the connection between UnCAL and the extended $\lambda\mathbf{G}$ -calculus has been theoretically and concretely more precisely implemented.

The functional language FUnCAL (Matsuda and Asada, 2015) gives an efficient operational semantics of $\lambda\mathbf{G}$. Rather than equational theories, FUnCAL focuses on evaluating and manipulating UnCAL programs as usual functional programs. The syntax of FUnCAL is essentially the same as $\lambda\mathbf{G}$ and an abstract machine for FUnCAL, which respects the axioms of the extended $\lambda\mathbf{G}$ -calculus (hence it is a correct implementation), has been designed and implemented. The implementation is available at

<https://bitbucket.org/kztk/funcal>

It is based on Nakata and Hasegawa (2009)’s semantics of call-by-need computations, which extends Launchbury (1993)’s natural semantics of lazy evaluation with the black hole (Ariola and Blom, 1997; Ariola and Klop, 1996). With an embedded implementation of the lazy semantics inspired by Fischer et al. (2011) who used a monad to describe deferred computation in a certain lazy semantics, it is shown to be efficient about 4 to 6 times faster in concrete examples than an ordinary implementation of UnCAL (Matsuda and Asada, 2015, Table 1). It can be regarded as effectiveness of our semantic and axiomatic framework.

7. Discussion: Variations of UnCAL and Related Graph Calculi

The categorical structure we have clarified leads us naturally to other variations and to relate the UnCAL with other graph calculi, by modifying the current axioms AxG. In this section, we discuss variations of UnCAL and related graph calculi.

- *Ordered branching UnCAL*

In the original UnCAL considered in this paper, the branch of a tree has no order, i.e. \cup is commutative and idempotent, which is due to the bisimulation equivalence. The modification to ordered branches is pursued in (Hidaka et al., 2013a; Asada et al., 2013), where elaborated adaptations of the bisimulation and graph-theoretic semantics of structural recursion for ordered branches are developed. In our case setting, the modification is simpler. Ordered branch UnCAL can be axiomatised just by dropping the axioms marked “ \star ” in AxG of Fig. 6. Correspondingly, the λ G-calculus for ordered branches is obtained by dropping the axioms marked “ \star ” in the axioms of λ G. The ordered variant of Thm. 5.1 can be proved straightforwardly by accordingly modifying bisimulation as in (Hidaka et al., 2013a; Asada et al., 2013), the axioms and the proof of (Bloom and Ésik, 1993; Bloom et al., 1993) or (Salomaa, 1966; Milner, 1984) to deal with ordered branches.

- *Why not monoidal, rather than cartesian? — forbidding copy of graphs*

To choose the axioms of monoidal product, instead of cartesian product, means to forbid copying of graphs in substitution. This setting is more common in modelling graphs, such as (Gibbons, 1995; Hasegawa, 1997a; Corradini and Gadducci, 1999). Degenerated bialgebras in a strict monoidal category (known as a PROP (Mac Lane, 1965)) are used by Fiore and Campos (2013) to model directed acyclic graphs. That work was originated by Milner’s question (Fiore and Campos, 2013, Introduction) concerning extensions of his *bigraphs* (Milner, 2005). A similar approach can also be found in Plotkin’s algebraic modelling of bigraphs (Plotkin, 2012). Exploring a link to bigraphs would be interesting.

- *Why not traced, rather than iteration category? — cyclic sharing theories*

Dropping the axiom (CI), it no longer forms an iteration category, but still is a *traced* cartesian category (Joyal et al., 1996). Hasegawa clarified that to model graphs in calculi using a letrec-style notation, both linear and non-linear treatments of constructs are necessary. This clarification led him to *cyclic sharing theories* and its semantics: cartesian-center traced monoidal categories (Hasegawa, 1997b,a). It was also used in (Hasegawa, 1997a) to model Milner’s *action calculi* (Milner, 1996). These calculi and models may be useful for a variation of UnCAL.

- *How about premonoidal trace? — “arrows” and loops in Haskell*

The notion of trace operator can also be adapted to a further-weakened setting called premonoidal categories (Power and Robinson, 1997). It has been used to model recursive computation with side-effects (Benton and Hyland, 2003), and has been implemented as a feature called arrows with loops (Paterson, 2001) in Haskell. Modifying our axioms to referring the axioms of premonoidal trace (or arrows with loops) may be practically interesting. Here the idea of cyclic terms and arrow interpretation (Hamana, 2012) may be incorporated.

- *Why not closed? — higher-order types*

Introducing *cartesian closed* (or monoidal closed) axioms (Lambek and Scott, 1986), we obtain a higher-order version of UnCAL. It will be related to other higher-order graph calculi, such as the cyclic λ -calculus (Ariola and Blom, 1997) and higher-order cyclic sharing theories (Hasegawa, 1997a). Moreover, as a framework of equational logic for

graphs and structural/primitive recursion, introducing higher-order types is a natural choice, as we have done in the λG -calculus. But our consideration of the λG -calculus is restrictive than the cyclic λ -calculus, etc., because the λG -calculus has only first-order fixed point operator.

Making these connections explicit and such various ideas of variations and extensions are a powerful benefit of category theoretic characterisation, because category theory is a general language to bridge different worlds of mathematics and computational structures.

Acknowledgments We are grateful to Zhenjiang Hu and the members of the Bidirectional Graph Transformation Project at NII for various discussions on topics related to UnQL/UnCAL. We also acknowledge the stimulated discussion in the meeting of the Cooperative Research Project “Logical Approach to Metaprogramming” (2013-2015) at RIEC, Tohoku University. We are especially grateful to Atsusi Ohori for his comments on relevance to object-oriented database. This work was supported in part by JSPS KAK-ENHI Grant Number 24300001 and 25540002 (Hamana), 15K15966 (Matsuda), 23220001 and 15H05706 (Asada).

References

- Abramsky, S. and Jung, A. (1994). Domain theory. In *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Oxford University Press.
- Aczel, P., Adámek, J., Milius, S., and Velebil, J. (2003). Infinite trees and completely iterative theories: a coalgebraic view. *Theor. Comput. Sci.*, 300(1-3):1–45.
- Amadio, R. M. and Curien, P.-L. (1998). *Domains and lambda-calculi*. Cambridge University Press.
- Ariola, Z. M. and Blom, S. (1997). Cyclic lambda calculi. In *Theoretical Aspects of Computer Software, LNCS 1281*, pages 77–106.
- Ariola, Z. M. and Klop, J. W. (1996). Equational term graph rewriting. *Fundam. Inform.*, 26(3/4):207–240.
- Asada, K., Hidaka, S., Kato, H., Hu, Z., and Nakano, K. (2013). A parameterized graph transformation calculus for finite graphs with monadic branches. In *Proc. of PPDP '13*, pages 73–84.
- Baader, F. and Nipkow, T. (1998). *Term Rewriting and All That*. Cambridge University Press.
- Benton, N. and Hyland, M. (2003). Traced premonoidal categories. *Theoretical Informatics and Applications*, 37(4):273–299.
- Blanqui, F. (2000). Termination and confluence of higher-order rewrite systems. In *Rewriting Techniques and Application (RTA 2000)*, LNCS 1833, pages 47–61. Springer.
- Blanqui, F., Jouannaud, J.-P., and Okada, M. (2002). Inductive data type systems. *Theoretical Computer Science*, 272:41–68.
- Bloom, S. L. and Ésik, Z. (1993). *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer.
- Bloom, S. L. and Ésik, Z. (1994). Solving polynomial fixed point equations. In *Proc. of MFCS'94*, LNCS 841, pages 52–67.

- Bloom, S. L., Ésik, Z., and Taubner, D. (1993). Iteration theories of synchronization trees. *Inf. Comput.*, 102(1):1–55.
- Buneman, P. (2015). Database and programming: Two subjects divided by a common language? (invited talk). POPL’15.
- Buneman, P., Davidson, S., Hillebrand, G., and Suciu, D. (1996). A query language and optimization techniques for unstructured data . In *Proc. of ACM-SIGMOD’96*.
- Buneman, P., Davidson, S. B., Fernandez, M. F., and Suciu, D. (1997). Adding structure to unstructured data. In *Proc. of ICDT ’97*, pages 336–350.
- Buneman, P., Fernandez, M. F., and Suciu, D. (2000). UnQL: A query language and algebra for semistructured data based on structural recursion. *VLDB J.*, 9(1):76–110.
- Coquand, T. (1992). Pattern matching with dependent types. In *Proc. of the 3rd Work. on Types for Proofs and Programs*.
- Corradini, A. and Gadducci, F. (1999). An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7(4):299–331.
- Crole, R. (1993). *Categories for Types*. Cambridge Mathematical Textbook.
- Ésik, Z. (1999a). Axiomatizing iteration categories. *Acta Cybernetica*, 14:65–82.
- Ésik, Z. (1999b). Group axioms for iteration. *Inf. Comput.*, 148(2):131–180.
- Ésik, Z. (2000). Axiomatizing the least fixed point operation and binary supremum. In *Proc. of Computer Science Logic 2000*, LNCS 1862, pages 302–316.
- Ésik, Z. (2002). Continuous additive algebras and injective simulations of synchronization trees. *J. Log. Comput.*, 12(2):271–300.
- Fiore, M. P. and Campos, M. D. (2013). The algebra of directed acyclic graphs. In *Computation, Logic, Games, and Quantum Foundations*, LNCS 7860, pages 37–51.
- Fischer, S., Kiselyov, O., and Shan, C. (2011). Purely functional lazy nondeterministic programming. *J. Funct. Program.*, 21(4-5):413–465.
- Ghani, N., Lüth, C., and Marchi, F. D. (2005). Monads of coalgebras: rational terms and term graphs. *Mathematical Structures in Computer Science*, 15(3):433–451.
- Gibbons, J. (1995). An initial-algebra approach to directed acyclic graphs. In *MPC’95*, pages 282–303.
- Hagino, T. (1987). A typed lambda calculus with categorical type constructors. In *Category Theory in Computer Science’87*, LNCS 283, pages 140–157. Springer-Verlag.
- Hamana, M. (2010). Initial algebra semantics for cyclic sharing tree structures. *Logical Methods in Computer Science*, 6(3).
- Hamana, M. (2012). Correct looping arrows from cyclic terms: Traced categorical interpretation in Haskell. In *Proc. of FLOPS’12*, LNCS 7294, pages 136–150.
- Hamana, M. (2015). Iteration algebras for unql graphs and completeness for bisimulation. In *Proc. of Fixed Points in Computer Science (FICS’15)*, Electronic Proceedings in Theoretical Computer Science 191, pages 75–89.
- Hamana, M. (2016). Strongly normalising cyclic data computation by iteration categories of second-order algebraic theories. In *Proc. of Formal Structures for Computation and Deduction (FSCD’16)*. to appear in the Leibniz International Proceedings in Informatics (LIPIcs).
- Hasegawa, M. (1997a). *Models of Sharing Graphs: A Categorical Semantics of let and letrec*. PhD thesis, University of Edinburgh. Distinguished Dissertation Series, Springer-Verlag, 1999.

- Hasegawa, M. (1997b). Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi. In *Proc. of TLCA '97*, pages 196–213.
- Hidaka, S., Asada, K., Hu, Z., Kato, H., and Nakano, K. (2013a). Structural recursion for querying ordered graphs. In *Proc. of ACM SIGPLAN ICFP'13*, pages 305–318.
- Hidaka, S., Hu, Z., Inaba, K., and Kato, H. (2013b). GRoundTram: An integrated framework for developing well-behaved bidirectional model transformations. *Progress in Informatics*, 10.
- Hidaka, S., Hu, Z., Inaba, K., Kato, H., Matsuda, K., and Nakano, K. (2010). Bidirectionalizing graph transformations. In *Proc. of ICFP 2010*, pages 205–216.
- Hidaka, S., Hu, Z., Inaba, K., Kato, H., Matsuda, K., Nakano, K., and Sasano, I. (2011). Marker-directed optimization of uncal graph transformations. In *Proc. of LOPSTR'11*, pages 123–138.
- Immerman, N. (1987). Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778.
- Joyal, A., Street, R., and Verity, D. (1996). Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468.
- Klop, J. (1980). *Combinatory Reduction Systems*. PhD thesis, CWI, Amsterdam. volume 127 of Mathematical Centre Tracts.
- Klop, J., Oostrom, V., and Raamsdonk, F. (1993). Combinatory reduction systems: Introduction and survey. *Theor. Comput. Sci.*, 121(1&2):279–308.
- Lambek, J. and Scott, P. (1986). *Introduction to Higher Order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press.
- Launchbury, J. (1993). A natural semantics for lazy evaluation. In *Proc. of POPL'93*, pages 144–154.
- Libkin, L. (1992). An elementary proof that upper and lower powerdomain constructions commute. *Bulletin of the EATCS*, pages 175–177.
- Mac Lane, S. (1965). Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40–106.
- Mac Lane, S. (1971). *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag.
- Matsuda, K. and Asada, K. (2015). Graph transformation as graph reduction: A functional reformulation of graph-transformation language UnCAL. Technical Report GRACE-TR 2015-01, National Institute of Informatics. Available at http://grace-center.jp/rsc_tr-html
- Meertens, L. G. L. T. (1992). Paramorphisms. *Formal Asp. Comput.*, 4(5):413–424.
- Milner, R. (1984). A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466.
- Milner, R. (1989). *Communication and concurrency*. Prentice Hall.
- Milner, R. (1996). Calculi for interaction. *Acta Inf.*, 33(8):707–737.
- Milner, R. (2005). Axioms for bigraphical structure. *Mathematical Structures in Computer Science*, 15(6):1005–1032.
- Milner, R. (2006). Pure bigraphs: Structure and dynamics. *Inf. Comput.*, 204(1):60–122.
- Nakata, K. and Hasegawa, M. (2009). Small-step and big-step semantics for call-by-need. *J. Funct. Program.*, 19(6):699–722.

- Nishimura, S. and Ohori, A. (1999). Parallel functional programming on recursively defined data via data-parallel recursion. *J. Funct. Program.*, 9(4):427–462.
- Nishimura, S., Ohori, A., and Tajima, K. (1996). An equational object-oriented data model and its data-parallel query language. In *Proc. of ACM SIGPLAN Object-Oriented Programming Systems, Languages & Applications (OOPSLA '96)*, pages 1–17.
- Paterson, R. (2001). A new notation for arrows. In *Proceedings of ICFP'07*, pages 229–240.
- Plotkin, G. (2012). An algebraic view of bigraphs. A talk at Milner Symposium 2012, University of Edinburgh.
- Power, J. and Robinson, E. (1997). Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468.
- Salomaa, A. (1966). Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169.
- Sewell, P. M. (1995). *The Algebra of Finite State Processes*. PhD thesis, University of Edinburgh. Dept. of Computer Science technical report CST-118-95, also published as LFCS-95-328.
- Simpson, A. K. and Plotkin, G. D. (2000). Complete axioms for categorical fixed-point operators. In *Proc. of LICS'00*, pages 30–41.
- Staton, S. (2011). Relating coalgebraic notions of bisimulation. *Logical Methods in Computer Science*, 7(1).
- Winskel, G. (1993). *The Formal Semantics of Programming Languages*. The MIT Press.
- Yu, Y., Lin, Y., Hu, Z., Hidaka, S., Kato, H., and Montrieux, L. (2012). Maintaining invariant traceability through bidirectional transformations. In *Proc. of International Conference on Software Engineering, ICSE 2012*, pages 540–550.

Appendix A. The Graph Model of UnCAL

In this appendix, we reorganise the original graph theoretic definition of UnCAL graphs, bisimilarity, and constructors (Buneman et al., 2000) as an instance of our categorical model.

A.1. UnCAL graphs

We define the UnCAL graphs formally. Let L be a set of labels. An *UnCAL graph* G is a quadruple

$$(V, E, I, O)$$

where V is a set of *vertices*, $E \subseteq V \times (L \cup \{\varepsilon\}) \times V$ is a set of *edges*, and a function $I : X \rightarrow V$ and a relation $O \subseteq V \times Y$ are the connection of roots X to vertices, and that of vertices to leaves Y , respectively, where X and Y are sets of markers. A *finite* UnCAL graph is an UnCAL graph whose set of vertices is finite. We write the set of UnCAL graphs as $\mathbf{Gr}(Y, X)$.

Two UnCAL graphs are bisimilar if the infinite trees obtained by unfolding sharings and cycles are identical after short-cutting all the ε -edges. Formally, it is defined as

follows. We write $v \rightarrow^l u$ if there is an edge $(v, l, u) \in E$ between vertices $v, u \in V$ in an UnCAL graph $G = (V, E, I, O)$. An *extended simulation* \mathcal{X} between an UnCAL graph $G_1 = (V_1, E_1, I_1, O_1)$ and $G_2 = (V_2, E_2, I_2, O_2)$ is a relation satisfying the following conditions:

- (i) if $(v, u) \in \mathcal{X}$, for any path $v = v_0 \rightarrow^\varepsilon \cdots \rightarrow^\varepsilon v_n \rightarrow^l v_{n+1}$ where $n \geq 0$, there is a path $u = u_0 \rightarrow^\varepsilon \cdots \rightarrow^\varepsilon u_m \rightarrow^l u_{m+1}$ where $m \geq 0$ and $(v_{n+1}, u_{m+1}) \in \mathcal{X}$,
- (ii) if $(v, u) \in \mathcal{X}$, for any path $v = v_0 \rightarrow^\varepsilon \cdots \rightarrow^\varepsilon v_n$ where $n \geq 0$ such that $(v_n, x) \in O_1$, there is a path $u = u_0 \rightarrow^\varepsilon \cdots \rightarrow^\varepsilon u_m$ where $m \geq 0$ such that $(u_m, x) \in O_2$.

Moreover, \mathcal{X} is called *extended bisimulation* if its opposite relation is also an extended simulation relation between G_2 and G_1 . Two UnCAL graphs $G_1, G_2 \in \mathbf{Gr}(Y, X)$ are *extended bisimilar*, denoted by $G_1 \sim G_2$, if there is an extended bisimulation \mathcal{X} between G_1 and G_2 such that $(I_1(x), I_2(x)) \in \mathcal{X}$ for any $x \in X$. We define $\mathbf{Gr}(Y, X)$ to be the quotient of $\mathbf{Gr}(Y, X)$ by the extended bisimilarity.

A.2. The category of UnCAL graphs and a model

UnCAL graphs form a category \mathbf{Gr} whose objects are sets of markers, and whose hom-sets are $\mathbf{Gr}(Y, X)$. Namely, a morphism $G \in \mathbf{Gr}(Y, X)$ is an UnCAL graph (up to extended bisimilarity) with roots X and leaves Y . It is clear that the following definitions are all well-defined with respect to extended bisimilarity. For an object X , the identity morphism is

$$(X, \emptyset, \text{id}_X, \Delta_X)$$

where Δ_X is the diagonal relation on X . For $G_1 \in \mathbf{Gr}(Y, X)$ and $G_2 \in \mathbf{Gr}(Z, Y)$, their composition $G_1 \circ G_2 \in \mathbf{Gr}(Z, X)$ is

$$\begin{aligned} (V, E, I_1, O_2) \text{ where } & (V_1, E_1, I_1, O_1) = G_1 \\ & (V_2, E_2, I_2, O_2) = G_2 \\ & V = V_1 \cup V_2 \\ & E = E_1 \cup E_2 \cup \{(v, \varepsilon, I_2(x)) \mid (v, x) \in O_1\} \end{aligned}$$

We show that \mathbf{Gr} is an iteration category. \mathbf{Gr} is cartesian: for X_1 and X_2 , their product is the disjoint union $X_1 + X_2$ (with some canonical renaming of markers). The i -th projection $\pi_i \in \mathbf{Gr}(X_1 + X_2, X_i)$ is

$$(X_i, \emptyset, \text{id}_{X_i}, \{(x, x) \in X_i \times (X_1 + X_2) \mid x \in X_i\}).$$

For $G_i \in \mathbf{Gr}(Y, X_i)$, their pairing morphism $\langle G_1, G_2 \rangle \in \mathbf{Gr}(Y, X_1 + X_2)$ is

$$\begin{aligned} & (V_1 \cup V_2, E_1 \cup E_2, I_1 \cup I_2, O_1 \cup O_2) \\ & \text{where } (V_1, E_1, I_1, O_1) = G_1 \\ & \quad (V_2, E_2, I_2, O_2) = G_2 \end{aligned}$$

where we rename elements in V_1 and V_2 so that they are disjoint. The terminal object is the empty set \emptyset , and the unique morphism $\langle \rangle_X \in \mathbf{Gr}(X, \emptyset)$ is

$$(\emptyset, \emptyset, \emptyset, \emptyset).$$

The category \mathbf{Gr} has a Conway operator satisfying the commutative identity: For $G \in \mathbf{Gr}(Y + X, X)$, $(G)^\dagger \in \mathbf{Gr}(Y, X)$ is

$$\begin{aligned} & (V, E', I, \{(v, y) \in O \mid y \in Y\}) \\ & \text{where } (V, E, I, O) = G \\ & E' = E \cup \{(v, \varepsilon, I(x)) \mid (v, x) \in O\} \end{aligned}$$

It is clear that the above structures satisfy the axioms of an iteration category. The Fig. 7 and Fig. 8 actually show how the axioms \mathbf{AxG} are interpreted in the category \mathbf{Gr} of UnCAL graphs.

Next, we show that $\{\&\}$ has an L -monoid structure in \mathbf{Gr} . First, it has a monoid structure: the unit $\eta \in \mathbf{Gr}(\emptyset, \{\&\})$ is

$$(\{\&\}, \emptyset, \{\& \mapsto \&\}, \emptyset)$$

and the multiplication $\mu \in \mathbf{Gr}(\{\&\} + \{\&\}, \{\&\})$ is

$$(\{\&\}, \emptyset, \{\& \mapsto \&\}, \{(\&, \&_1), (\&, \&_2)\})$$

where we renamed $\{\&\} + \{\&\}$ to $\{\&_1, \&_2\}$. It is clear that this satisfies the axioms of a commutative monoid and the axiom $(\mu)^\dagger = \text{id}$. Finally, for $\ell \in L$

$$\llbracket \ell \rrbracket_L^{\{\&\}} = (\{v, u\}, \{(v, \ell, u)\}, \{\& \mapsto v\}, \{(u, \&)\}).$$

Thus, we have an L -monoid $(\{\&\}, \llbracket - \rrbracket_L^{\{\&\}})$, i.e., a model of pure theory in \mathbf{Gr} .

The induced categorical interpretation $\llbracket - \rrbracket_{\mathbf{Gr}}^{\{\&\}}$ provides the graph theoretic meaning of a term. For example,

$$\llbracket \mathbf{a} : (\{\mathbf{b} : x\} \cup \{\mathbf{c} : x\}) \diamond \text{cycle}(x \triangleleft \mathbf{d} : (\{\mathbf{p} : y_1\} \cup \{\mathbf{q} : y_2\} \cup \{\mathbf{r} : x\})) \rrbracket_{\mathbf{Gr}}^{\{\&\}} = \sim$$

Each UnCAL graph in Fig. 2 is exactly the interpretation of the corresponding term by $\llbracket - \rrbracket_{\mathbf{Gr}}^{\{\&\}}$. The model $(\{\&\}, \llbracket - \rrbracket_L^{\{\&\}})$ in \mathbf{Gr} is a sound and complete model of \mathbf{AxG} as shown in Theorem 5.7.

(Mark)	$\frac{Y = \langle\langle y_1, \dots, y_n \rangle\rangle}{Y \vdash y_{i_Y} : \&} \mapsto \vdash \pi_i : \mathbf{G}^{ Y } \rightarrow \mathbf{G}$
(Emp)	$\frac{}{Y \vdash ()_Y : \langle\langle \rangle\rangle} \mapsto \vdash \lambda y.() : \mathbf{G}^{ Y } \rightarrow \mathbf{1}$
(Nil)	$\frac{}{Y \vdash \{\}_Y : \&} \mapsto \vdash \lambda y.\{\} : \mathbf{G}^{ Y } \rightarrow \mathbf{G}$
(Man)	$\frac{}{y_1, y_2 \vdash \lambda \langle\langle y_1, y_2 \rangle\rangle : \&} \mapsto \vdash \cup : \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$
(Com)	$\frac{Y \vdash s : Z}{X \vdash s \diamond t : Z} \mapsto \vdash g : \mathbf{G}^{ Y } \rightarrow \mathbf{G}^{ Z }$ $\frac{X \vdash t : Y}{X \vdash s \diamond t : Z} \mapsto \vdash h : \mathbf{G}^{ X } \rightarrow \mathbf{G}^{ Y }$ $\frac{}{X \vdash s \diamond t : Z} \mapsto \vdash g \circ h : \mathbf{G}^{ X } \rightarrow \mathbf{G}^{ Z }$
(Label)	$\frac{\ell \in L}{Y \vdash t : \&} \mapsto \vdash \llbracket \ell \rrbracket_L^{\mathbf{G}} : \mathbf{G} \rightarrow \mathbf{G}$ $\frac{Y \vdash t : \&}{Y \vdash \ell : t : \&} \mapsto \vdash g : \mathbf{G}^{ Y } \rightarrow \mathbf{G}$ $\frac{}{Y \vdash \ell : t : \&} \mapsto \vdash \llbracket \ell \rrbracket_L^{\mathbf{G}} \circ g : \mathbf{G}^{ Y } \rightarrow \mathbf{G}$
(Pair)	$\frac{Y \vdash s : X_1}{Y \vdash t : X_2} \mapsto \vdash g : \mathbf{G}^{ Y } \rightarrow \mathbf{G}^{ X_1 }$ $\frac{Y \vdash t : X_2}{Y \vdash \langle s, t \rangle : X_1 + X_2} \mapsto \vdash h : \mathbf{G}^{ Y } \rightarrow \mathbf{G}^{ X_2 }$ $\frac{}{Y \vdash \langle s, t \rangle : X_1 + X_2} \mapsto \vdash \lambda y^{\mathbf{G}^{ Y }}. (g \ y, h \ y) : \mathbf{G}^{ Y } \rightarrow \mathbf{G}^{ X_1 } \times \mathbf{G}^{ X_2 }$
(Cyc)	$\frac{Y + X \vdash t : X}{Y \vdash \text{cycle}^X(t) : X} \mapsto \vdash g : \mathbf{G}^{ Y } \times \mathbf{G}^{ X } \rightarrow \mathbf{G}^{ X }$ $\frac{}{Y \vdash \text{cycle}^X(t) : X} \mapsto \vdash \lambda y^{\mathbf{G}^{ Y }}. \text{fix}(\lambda x^{\mathbf{G}^{ X }}. g \ (y, x)) : \mathbf{G}^{ Y } \rightarrow \mathbf{G}^{ X }$
(Def)	$\frac{Y \vdash t : \&}{Y \vdash (x \triangleleft t) : x} \mapsto \vdash g : \mathbf{G}^{ Y } \rightarrow \mathbf{G}$ $\frac{}{Y \vdash (x \triangleleft t) : x} \mapsto \vdash g : \mathbf{G}^{ Y } \rightarrow \mathbf{G}$

In (Mark), if $n = 1$ then the interpretation is $\lambda x.x : \mathbf{G} \rightarrow \mathbf{G}$.

Fig. 12. Translation $\llbracket - \rrbracket$ from UnCAL terms to $\lambda\mathbf{G}$ terms